VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Odhad a sledování póz netexturovaných objektů

Pose Estimation and Tracking of Untextured Objects

2018 Aleš Togner

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Aleš Togner**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Odhad a sledování póz netexturovaných objektů
Pose Estimation and Tracking of Untextured Objects

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je naimplementovat algoritmus pro detekci a určení pózy zvoleného objektu bez nápadné textury. Konkrétně uvažovanou situací je nalezení a následné sledování objektu předem známého tvaru v obraze v reálném čase. Výstupní obraz bude doplněn o model sledovaného objektu tak, aby co nejvěrněji kopíroval polohu sledovaného objektu.

1. Seznamte se s metodami pro sledování póz objektů ve videosekvencích.
2. Vybranou metodu naimplementujte.
3. Funkčnost metody demonstrujte na vhodných videosekvencích.
4. Zhodnoťte dosažené výsledky, a to zejména s ohledem na robustnost vůči zákrytům sledovaného objektu.
5. Zvolené postupy a dosažené výsledky pečlivě zdokumentujte v textu práce.

Seznam doporučené odborné literatury:

[1] Hinterstoisser, S., et al., Gradient Response Maps for Real-Time Detection of Texture-Less Objects, TPAMI 2012.
[2] Hsiao, E., Hebert, M. Gradient Networks: Explicit Shape Matching Without Extracting Edges. AAAI 2013.
[3] Hsiao, E., Hebert, M. Occlusion Reasoning for Object Detection under Arbitrary Viewpoint. CVPR 2012.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Tomáš Fabián, Ph.D.**

Datum zadání: 01.09.2016
Datum odevzdání: 30.04.2018

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 30. dubna 2018 .....

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.

V Ostravě 30. dubna 2018 .....

Rád bych na tomto místě poděkoval vedoucímu práce Ing. Tomáši Fabiánovi, Ph.D. za užitečné rady a kontrolu postupu práce.

Abstrakt

Cílem mé diplomové práce bylo naimplementování algoritmu pro detekci a určení pózy netexturovaného, předem známeho objektu v reálném čase včetně grafického výstupu. Požadavkem rovněž bylo použití barevných i hloubkových dat. Analýza současného stavu se zabývá přehledem metod řešících tuto problematiku, ze kterých jsem na základě požadavků vybral a naimplementoval detekční pipeline založenou na algoritmech LINEMOD a Iterative Closest Point. Tyto metody rovněž byly více přiblíženy v teoretické části. Testování proběhlo na vlastních, složitějších objektech než v původní práci, zejména z hlediska zákrytu objektu a horší kvality hloubkových dat. Rovněž proběhl test oproti algoritmu LINE-2D, který využívá pouze barevných informací. Detekcí bylo úspěšně dosaženo ve stanoveném čase a také byly přiblíženy některé nedostatky zvoleného přístupu.

Klíčová slova: počítačové vidění, detekce objektů, odhad pózy, RGB-D, LINEMOD, Iterative Closest Point

Abstract

The goal of my diploma thesis was to implement an algorithm for detection and pose estimation of a known, textureless object in real time including a graphical output. The use of both color and depth data was also a requirement. The state-of-the-art analysis gives an overview of methods that solve this problem, from which I picked one based on the requirements and implemented an object detection pipeline based on the LINEMOD and Iterative Closest Point algorithms. Testing was mainly done on own, more complex objects than in the original work, especially in terms of robustness to occlusion and poor quality of the depth image. A test against the LINE-2D algorithm, which uses only color data, was also performed. The detections were successfully achieved at required speed and some flaws of the chosen approach were also revealed.

Key Words: computer vision, object detection, pose estimation, RGB-D, LINEMOD, Iterative Closest Point

Obsah

Seznam použitých zkratek a symbolů

FPS	–	Frames per second, snímky za sekundu
GPU	–	Graphic Processing Unit
AR	–	Augmented reality
h.o.t.	–	Higher Order Terms
ICP	–	Iterative Closest Point
SVM	–	Support Vector Machine
IDE	–	Integrated Development Environment

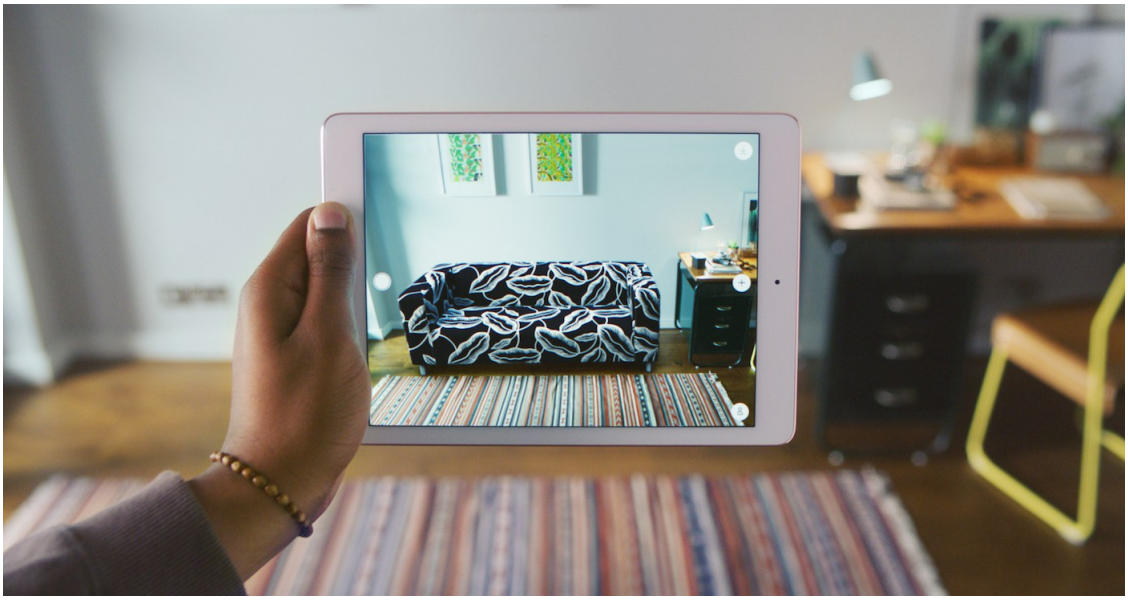
Seznam obrázků

Seznam tabulek

1 Úvod

Cílem mé diplomové práce je problematika detekce a sledování předem známého, netexturovaného objektu ve scéně a následný odhad jeho pózy. V první teoretické části práce se věnuji analýze této problematiky a již dostupným metodám jejího řešení. V části druhé více přiblížím jednu ze zmíněných metod, třetí část práce je věnována samotné implementaci zvolené metody a část poslední obsahuje dosažené výsledky a závěr.

S vývojem výpočetní technologie došlo k otevření nových možností pro její využití, a to nejen díky neustálému nárůstu výkonu, ale i z důvodu její miniaturizace a poklesu ceny, což vedlo k jejímu zpřístupnění širšímu rozsahu uživatelů a využití pro nové účely. Jednou z těchto oblastí je počítačové vidění, které je využito pro nejrůznější účely, např. pro bezpečnost (detekce osob na bezpečnostní kameře nebo sledování aktivity řidiče v autě), automatizaci činnosti robota nebo i zábavu, kde je dobrým příkladem kamera Kinect od společnosti Microsoft ve spolupráci se společností PrimeSense. Ta je vybavena jak standardní kamerou, tak i hloubkovým senzorem, což rozšiřuje obraz RGB na RGB-D. V kombinaci s velmi příznivou cenou tak zařízení našlo uplatnění i v oblasti počítačového vidění, a to jak v akademickém výzkumu, tak i u amatérských vývojářů. Základním úkolem počítačového vidění je detekce daných objektů na základě jeho vlastností, jako je barva nebo tvar. Ty určují příznaky zvolené tak, aby splňovaly určité požadavky - co nejrychlejší výpočet příznaků a na jejich základě porovnávání prvků v obraze, jednoznačné rozlišení objektů a současně robustnost vůči rušivým elementům, jako je snížená kvalita obrazu nebo částečné zakrytí objektu. Rozšíření barevných kanálů RGB o hloubkou informaci kanálu D poté rozšiřuje možnosti příznaků, např. rekonstrukcí normálových vektorů nebo určením 3D pozice daného bodu. Výstupní možnosti algoritmu se poté rovněž posouvají z 2D pozice v obraze do 3D a lze rovněž určit rotaci objektu kolem jeho tří os. Jedná se tedy o určení pózy objektu s šesti stupni volnosti (ang. degrees of freedom - DoF). To má využití např. v rozšířené realitě (augmented reality), kde se do obrazu reálného světa přidávají prvky vykreslené počítačem, jejichž umístění na daném objektu je určeno jeho 6 DoF pozicí. Rozšířená realita našla uplatnění u široké veřejnosti zejména zkoušením produktů, příkladem je mobilní aplikace Ikea Place od firmy IKEA, která umožňuje umístění nábytku do prostoru na obrazovce chytrého telefonu. Lze tak tedy zjistit, zda se křeslo nebo skříň hodí do místnosti bez nutnosti jejich zakoupení nebo transportace. Podobné využití nabízí společnost Lenskart pro zkoušení brýlí a firma ModiFace vyvinula aplikace na zkoušení kosmetiky pro firmy jako Urban Decay, L'Oréal nebo Avon Beauty.



Obrázek 1: Aplikace IKEA Place. Dostupné z: https://digiday.com/marketing/ikea-wants-consolidate-three-mobile-apps-one/

Vzhledem k vysokým hardwarovým nárokům v oblasti her zatím AR nenašla široké využití, firma Niantic však kombinací technologie Google Maps s celosvětově známými Pokémony vyvinuli hru pro iOS a Android Pokémon Go, která dosáhla před koncem roku 2016 více než 500 miliónů stažení, lze tedy očekávat, že s narůstající výpočetní silou se objeví mnoho dalších podobných her. S vývojem v oblasti hardwaru jdou kupředu i algoritmy, počítačové vidění je tedy používáno k postupně komplexnějším a nebezpečnějším činnostem jako je řízení auta. V této oblasti experimentují firmy Waymo, patřící firmě Google a Tesla, která již nabízí autonomně řízená auta běžným zákazníkům, je však nutno podotknout, že navzdory oficiálnímu jménu Autopilot se nejedná o stoprocentně spolehlivý systém. Vytvoření algoritmu, který detekci provádí přesně a současně v reálném čase, což je pro některé oblasti využití nezbytností, tedy není snadným úkolem.

2 Současný stav

Metody detekce a sledování objektů v obraze a odhadu jeho pózy lze rozdělit podle několika kritérií v závislosti na přístupu k problému.

První z těchto rozdělení je na řídké příznakové metody (sparse feature based), husté metody (dense) a metody srovnávání se vzory (template matching). V prvním případě detekce probíhá na základě extrakce zajímavých bodů (často invariantních vůči škále), jejich popisu lokálními deskriptory (často afinními a invariantními vůči osvětlení) a následným porovnáváním oproti databázi. Přístup dosahuje dobrých výsledků u texturovaných objektů, s netexturovanými objekty však má problémy kvůli nedostatku diskriminačních příznaků. Alternativou řídkého přístupu je přístup hustý, kde se každý pixel podílí na predikci výstupu. Příkladem je zobecněné Houghovo hlasování, kde všechny pixely hlasují v kvantizovaném prostoru predikcí (např. 2D pozice středu objektu a škála) a pixel s nejvíce hlasy se považuje za vítěze. [12] V posledním případě se srovnávají části obrazu přímo oproti vzorům, které zachycují objekt z různých pozic a vzdáleností. Pravděpodobnost výskytu daného vzoru na určité pozici poté určuje míra podobnosti. Tento přístup je vhodnější pro netexturované objekty, navíc lze využít vzorového obrazu pro další účely, např. určení záchytných bodů objektu pro robota. [5]

Druhé rozdělení je podle přístupu k sledování objektu. V prvním případě probíhá zpracování obrazu nezávisle na obrazech předchozích, jedná se o tzv. sledování detekcí (tracking by detection), pohyb objektu poté lze určit z výsledků jednotlivých obrazů. V případě druhém se sledování realizuje na základě předchozích obrazů (tracking based approach). Další možné kritérium rozdělení je rychlost algoritmu, kde rozdělujeme přístupy v závislosti na tom, zda detekci provádějí v reálném čase, či nikoli. Za dolní hranici reálného času je považována 1 FPS. [6]

Vzhledem k množství existujících metod zde však nebude možné přiblížit všechny, v následujících odstavích tedy budou zmíněny především ty aktuální.

2.1 Template matching

Z oblastí, které pohánějí v této oblasti vývoj, jsou výpočetně výkonné algoritmy zapotřebí především v robotice, jelikož autonomní systémy se musí učit rozpoznávat nové objekty a opakovaně přizpůsobovat neznámému a proměnlivému prostředí. Pro aplikace s takovým důrazem na výpočetní čas je template matching v reálném čase přitažlivý z důvodu jednoduchého a rychlého učení nových objektů za běhu na rozdíl od statistických metod. Důvodem je přístup těchto metod, jejichž učení se zaměřuje na rozpoznávání neznámých objektů podle určitých tříd oproti rozpoznávání předem známých instancí objektů z různých míst pohledu. Stále se však nejedná o triviální problém, jelikož data se mezi trénovacími vzory a testovanou sekvencí obrazů značně mění v osvětlení, zákrytu a bodu pohledu. Pokud je objekt dostatečně texturovaný pro nalezení klíčových bodů a na jejich základě identifikován, problém lze vyřešit za použití rychle vypočitatelných deskriptorů pro charakterizaci objektu. Tento přístup však selže na objektech netexturovaných, jejichž vzhled je definován především tvarem obrysu. [1]

Pro vyřešení problému Hinterstoisser [1] navrhnul přístup založený na template matchingu pro instance pevných 3D objektů tak, aby se vzorové obrazy rychle vytvořily i porovnávaly a současně byl robustní vůči překrytí objektů, drobným posuvům a deformacím. Současně vzorové obrazy poskytují hrubý odhad pózy objektu, což je důležité hlavně pro roboty a jejich interakci s prostředím. Přístup kombinuje RGB gradienty objektu a normálové vektory vypočítané z hloubkového obrazu a dosahuje vysokých rychlostí jejich diskretizací, což umožňuje porovnávání za použitím pouze binárních operací a následnou linearizací, odtud název metody LINEMOD – linearized modalities. Přístup byl dále modifikován a optimalizován pro použití s 3D modelem hledaného objektu [2] a dosahoval vysokých přesností. Nevýhodou je absence invariance algoritmu vůči škále tělesa a lineární nárůst výpočetního času s počtem vzorových obrazů, přístup je tedy zejména vhodný pro sledování jednoho tělesa.

Algoritmus s lepším výpočetním časem vzhledem k počtu vzorů vyvinuli Cao, Yaser a Banerjee [5] použitím template matchingu na základě normalizované vzájemné korelace počítaném na GPU násobením matic. Při detekci patnácti objektů dosáhli rychlosti 20 FPS a přesnosti detekce srovnatelnou s LINEMODem, který potřeboval na detekci stejného počtu objektů přes sekundu na jeden obraz. Bylo tak však dosaženo za použití vysoce výkonné grafické karty (NVIDIA Quadro K6000 s 2880 jádry a 12 GB paměti), což silně omezuje použití algoritmu pro běžné účely. V případě jednoho objektu je algoritmus oproti přístupu LINEMOD zhruba čtyřikrát rychlejší (26,3ms na jeden obraz oproti 119ms), nelze tedy vyloučit, že za použití slabšího GPU v případě sledování jediného objektu by LINEMOD byl lepším řešením.

Rios-Cabrera and Tuytelaars [23] použili přístup LINEMOD pro 3D detekci a rozšířili jej dvěma způsoby: učením vzorů diskriminativním způsobem a kaskádovým schématem, které detekci urychluje. Diskriminativní učení bylo aplikováno třemi způsoby: zaprvé to znamená, že v každém vzorovém obraze se soustřeďuje na charakteristické vlastnosti (DTT - discriminatively trained templates). Dále, váhy vzorů jsou učeny a jejich skóre kombinována pomocí boostovacího schématu – v tomto kroku se tvoří kaskáda urychlující výpočet. Posledním vylepšením je vyladění vzorů obrazu vzhledem k počtu nenulových bitů. Z posledních dvou vylepšení vychází název metody DTT-OPT (DTT Optimized). Oproti LINEMODu bylo dosaženo mnohem lepší škálovatelnosti vzhledem k počtu objektů, na datasetu Hinterstoissera [2] za použití stejného postprocessingu a přístupu k tvorbě vzorů (nazváno DTT-3D) algoritmus prokázal skoro o řád rychlejší detekci, jelikož za současné detekce všech 15 objektů detekce proběhla dvakrát rychleji, než u jediného objektu u LINEMODu. Menší nevýhodou je však delší čas učení, jelikož kombinuje vzory s boostingem pro vytvoření silného klasifikátoru, který je poté konvertován do kaskády.

V rychlosti byl algoritmus DTT-3D dále překonán metodou rovněž založenou na LINEMODu nazvanou Hashmod [24], které dosáhla sublineární škálovatelnosti aplikací hashovacích funkcí na deskriptory obrazu. Algoritmus vyhledává pomocí sliding window zajímavé objekty, přičemž v každém místě na obraze je vypočten deskriptor x. Pokud je vzdálenost mezi x a nejbližším sousedem $x_{i,j}$ z množiny vzorových deskriptorů D dostatečně malá, místo v obraze

s vysokou pravděpodobností obsahuje objekt i v póze j. Toto nearest neighbour vyhledávání je provedeno právě pomocí hashování deskriptorů. Množina D se nejprve rozdělí do shluků D_s, kde s jsou podobné škály. To vede k použití s oken různých velikostí během testování, která extrahují deskriptory rovněž o různých velikostech. Pro zvýšení přesnosti detekce je ke každému oknu přiřazen předdefinovaný počet hashovacích funkcí, vztahujících se k náhodným, ale překrývajícím se podmnožinám D_s. Cílem těchto funkcí je indexace vstupu x do podmnožiny D nazvanou bucket, která je naplněna deskriptory z D se stejnou hashovou hodnotou, což omezuje vyhledávání nejbližšího souseda x pouze na daný bucket oproti prohledávání celé množiny vzorů D. Hashovací funkce $h(x)$ jsou navrženy tak, aby vrátily krátký binární řetězec o b bitech extrahovaných z vstupu x. Bity jsou vybrány tak, aby byli s co nejlepší přesností nalezeni nejbližší sousedé a současně tak, aby celkový počet vrácených vzorů byl co nejmenší. Experimentováno bylo s několika možnostmi: náhodný výběr bitů, výběr založený na pravděpodobnosti, výběr založený na stromové struktuře a výběr založený stromové struktuře s rozptylem bodů pohledu, což se prokázalo jako nejlepší řešení. I přes překonání rychlostí LINEMODu i DTT-3D však přesnost mírně zaostávala za oběma algoritmy.

2.2 Příznakové přístupy

Metody provádějící lokalizaci nebo rekonstrukci scény v reálném čase fungují na základě odhadu polohy kamery porovnáváním hloubkového obrazu oproti databázi již viděných hloubkových obrazů. Běžným přístupem k výše uvedenému problému je redukce výpočetních nároků pomocí extrakce zájmových bodů (interest points). Příkladem takové techniky je systém KinectFusion, který ukázal, že hloubkové senzory spolu s paralelizovanými programy běžících na výkonných grafických kartých dokáží v reálném čase provést sledování po jednotlivých snímcích a 3D rekonstrukci scény při hustých vstupních datech. Dostupné velikosti paměti však omezují rekonstrukci na malá prostředí, např. kanceláře. Pro pokrytí větších oblastí musí být scéna rozdělena do několika částí, které jsou zpracovány samostatně. Jedním z přístupů, jak tyto části znovu spojit je extrakce zájmových bodů, které poté mohou být rychle porovnávány. Určení vhodných bodů v hloubkových datech s šumem je však výpočetně náročné, v online systémech tedy má jen omezené využití.[18]

Jednou ze starších, ale neznámějších a stále používaných příznakových metod je SIFT (Scale Invariant Feature Transform) [30] což je, jak už napovídá název, metoda pro extrakci odlišných příznaků invariantních vůči škále a rotaci, které lze použít k spolehlivému matchingu mezi různými pohledy scény. Dále zprostředkovávají robustní detekci navzdory afinním transformacím, změnám osvětlení a bodu pohledu a šumu. Příznaky jsou vysoce rozlišující v tom smyslu, že jediný příznak lze s vysokou pravděpodobností správně identifikovat oproti velké databázi přiznaků z mnoha scén. To zprostředkovává bázi pro detekci objektu a scény. Výpočetní čas extrakce těchto příznaků je minimalizován kaskádovým filtrovacím přístupem, kde nákladnější operace jsou aplikovány pouze v lokacích, které prošly počátečním testem. Výpočet příznaků sestává z následujících kroků:

- Detekce extrémů napříč obrazem a škálami: Hledání potenciálních zájmových bodů invariantních vůči škále a orientaci napříč všemi škálami a místy v obrazech, které je implementováno funkcí rozdílu gaussiánů.

- Lokalizace klíčových bodů: Na každé kandidátní lokaci je pro určení lokace a škály přiřazen detailní model. Klíčové body jsou vybrány na základě jejch stability.

- Přiřazení orientací: Jedna nebo více orientací je přiřazena každému klíčovému bodu na základě lokálních směrů gradientů. Každá následující operace je vykonána na datech, které byly transformovány podle přiřazené orientace, škály a pozice každého příznaku, což zprostředkovává invarianci vůči těmto transformacím.

- Deskriptor klíčových bodů: Lokální gradienty obrazu jsou měřeny při vybraném měřítku v okolní každého klíčového bodu. Ty jsou transformovány do reprezentace, která umožňuje značnou lokální deformaci tvaru a změnu v osvětlení.

Důležitým aspektem tohoto přístupu je generování velkého množství příznaků, které hustě pokrývají obraz v plném rozsahu škál i pozic. Typický obrázek velikosti 500×500 pixelů vrátí přibližně 2000 stabilních příznaků, záleží však na zvolených parametrech a obsahu obrázku. Množství příznaků je důležité především pro rozpoznávání malých objektů na složitém pozadí, kde je pro spolehlivou identifikaci požadováno, aby byly nalezeny alespoň tři příznaky. Ty jsou extrahovány z množiny referenčních obrázků a uloženy do databáze, nový obrázek je poté rozpoznán individuálním porovnáním každého příznaku ze vstupního obrázku s těmi v databázi a kandidátní příznaky jsou nalezeny podle Euklidovy vzdálenosti vektorů příznaků. Deskriptory klíčových bodů jsou vysoce rozlišující, což umožňuje jedinému příznaku najít s vysokou pravděpodobností jeho protějšek v databázi. Současně to však znamená, že v obrázku s velkým množstvím informací nebude mít mnoho příznaků z pozadí žádný korektní protějšek v databázi, což má za důsledek mnoho nekorektních detekcí. Korektní detekce pak lze vyfiltrovat z celé množiny detekovaných klíčových bodů identifikací podmnožin, které mají shodný objekt, pozici, škálu a orientaci. Pravděpodobnost, že několik příznaků bude souhlasit s těmito parametry je o mnoho nižší, než že kterýkolik samostatný příznak bude nekorektně identifikován. Určení těchto konzistentních shluků lze rychle provést pomocí rychlé implementace zobecněné Houghovy transformace pomocí hash table. Každý shluk tří nebo více příznaků, které souhlasí s objektem i pózou je poté podrobněji verifikován. Nejprve je vytvořen odhad nejmenšího čtverce pro afinni aproximaci pózy objektu. Každé další příznaky konzistentní s touto pózou jsou identifikovány a odlehlé příznaky zahozeny. Dále je proveden detailní výpočet pravděpodobnosti, že určitá množina příznaků indikuje výskyt objektu při dané přesnosti odhadu a počtu potenciálně falešných detekcí. Objekty, které projdou všemi testy pak lze považovat s velkou pravděpodobností za korektní. Všechny kroky lze rychle implementovat, na procesoru Pentium 4 2GHz trvala detekce čtyř objektů v obrázku 64×315 pixelů méně než 0,3 sekundy.

Dalším detektorem a deskriptorem příznaků, který se objevil o pár let později je SURF (Speeded Up Robust Features) [34]. Podobně jako u výše uvedeného SURF, proces spočívá v nálezu zájmových bodů, které se nacházejí v rozlišujících lokacích v obrázku, jako např. rohy a rozbočení ve tvaru T. Na rozdíl od něj však generuje podstatně méně bodů. Nejcennější vlastnost takového detektoru bodů je opakovatelnost, t.j. nalezení stejných zájmových bodů pod různými pohledy. Okolí bodu je reprezentováno vektorem příznaků, který musí být robustní vůči šumu, detekčním chybám a geometrickým a fotometrickým deformacím. Vektory jsou porovnávány s z různými obrazy rovněž na základě vzdálenosti, např. Euklidovy nebo Mahalanobisovy vzdálenosti. Detektor je založen na Hessově matici a integrální reprezentaci obrazu, kde hodnota pixelu v bodě $b = (x, y)$ udává součet hodnot všech pixelů v obdélníkové oblasti tvořené počátečním pixelem $a = (0, 0)$ a pixelem b. Důležitou vlastností zde je, že výpočet sumy intenzit v libovolné obdélníkové oblasti lze provést čtyřmi operacemi sčítání, odtud pochází název detektoru Fast-Hessian. V testech dosáhl přibližně 10% větší přesnosti než předešlé metody včetně SIFT, kterou rovněž trojnásobně překonal rychlostí.

Holzer et al. [18] tento problém vyřešili aproximací response map detektorů zájmových bodů pomocí regresních stromů. Body jsou určeny podle zakřivení povrchu, vypočítaného z normálových vektorů povrchu v okolí daného bodu. Zvolené body jsou ty, na kterých je zakřivení dosahuje maxima a měly by splňovat následující čtyři kritéria:

- Sparseness (řídkost): ve scéně by měl být jen malý počet bodů, definováno prahovou hodnotou
- Repeatability (opakovatelnost): body by měly být identifikovány ze všech bodů pohledu, lze měřit rekonstrukcí dat
- Distinctiveness (jednoznačnost): oblast v okolí bodu by měla být unikátní, závisí na metodě matchingu, obtížná evaluace
- Efficiency (výkonnost): body by měly být určeny rychle

Chodu v reálném čase bylo dosaženo za použití 1 jádra 2,26GHz Intel Core 2 Quad a 4 GB RAM.

Bo, Ren a Fox vyvinuli nový přístup k algoritmu učení bez učitele Hierarchical Matching Pursuit (HMP), pracujícímu s RGB-D obrazy [16], založeném na slovníkovém algoritmu učení K-SVD a vlastnosti objektů reprezentovali řídkými kombinacemi klíčových slov. Původní HMP používalo pouze obrazy v šedé škále[17], což bylo v mnoha případech nedostačující. Cílem je naučení slovníku, což je sada vektorů (kódů), kde data jsou reprezentována řídkou, lineární kombinací záznamů ve slovníku, např. patch o velikosti 5×5 je vektor délky $5 \times 5 \times 8$, kde poslední komponenta se skládá z greyscale intenzity, RGB-D hodnot a 3D normály povrchu. Přístup K-SVD generalizuje K-Means s parametrem řídkosti (sparsity), který určuje počet nenulových parametrů, příčemž hodnota sparsity nastavená na 1 přesně reprodukuje K-Means. Testováním

na dostupných datasetech (Willow + challenge, 2D3D a jiné) bylo dosaženo konzistentně lepších výsledků v porovnáním s aktuálními metodami.

Payet a Todorovic [20] místo zájmových bodů použili jako základní příznak obrys objektu v metodě nazvané Bag of Boundaries (BOB). Obrysy byly vybrány z důvodu lepší invariance vůči barvě, textuře nebo změnám jasu oproti ostatním příznakům, což umožňuje značnou redukci velikosti trénovací množiny potřebné k dosažení vysoké úspěšnosti. Použití obrysů dovoluje řídkou reprezentaci objektu z více pohledů za pomocí několika vzorových tvarů. Lze je interpretovat jako mentální obrazy kategorie objektu, které jsou široce považovány za jeden z důležitých prvků lidského vidění. Příznak BOB vyskytující se v daném bodě obrazu je histogram hranic, což jsou obrysy vyskytující se v okolí BOB a patřící do popředí. V případě výskytu objektu budou jeho hranice pokryty mnoho BOB. Obraz a vzory jsou tedy reprezentovány deformovatelnými 2D svazy BOB, které spolu poskytují silnější podporu hypotézy výskytu oproti kterémukoliv individuálnímu obrysu. 3D rozpoznávání pak probíhá na základě porovnávání BOB obrazu a vzoru místo přímému porovnávání jednotlivých hran oproti vzorovým tvarům. Algoritmus since dobře generalizoval mezi objekty, ani zdaleka však nedosahoval reálného času a odhady 3D pózy nebyly příliš přesné.

Bonde et al. [15] vytvořili framework využívající jako vstup point cloud a multi class random forest s pro detekci objektů. Pro trénování byl využit scan objektu ze všech možných úhlů, který lze použít pro simulaci různých bodů pohledu. V každém zvoleném pohledu proběhla detekce edgeletů, které lze získat z poměrů hlavních křivostí pro každý vstupní bod pro vnitřní hrany a z oblastí s velkou diskontinuitou hloubky pro hrany vnější. Orientace bodů edgeletů byla dále kvantizována do 8 přihrádek a samotné body kvantizovány do voxelové mřížky. Dominantní orientace voxelu byla určena výběrem kvantizované orientace s největším počtem bodů v přihrádce. Trénink random forestu využívá metody soft labelling, kde se nejprve kvantizují body pohledu do 16 tříd (2× pitch, 2× roll, 4× yaw). Simulovaná póza by poté mohla být mezi dvěmi kvantizovanými třídami, je jí tedy přiřazen soft label vzhledem ke každé třídě na základě její rotace. Vzdálenost mezi třídami je maximalizována iterativním přístupem, který automaticky detekuje překrývající se vzory a následně je využije pro zvýšení vzdálenosti. To zvyšuje robustnost přístupu vůči velkému množství objektů ve scéně. Benchmark proběhl na datasetu Hinterstoissera [2] a vlastním, navrženém datasetu Desk3D. Na obou datasetech jejich přístup LINEMOD překonal, nevýhodou však je rychlost, která pro jeden objekt a jeden obraz byla přibližně 1,5 sekundy.

Pavlakos, Zhou et al. [27] zvolili přístup, který umožňuje velký rozsah detekovaných typů objektů, ať už texturovaných či nikoli, na základě obecné třídy objektu nebo specifické instance. Odhad 6 DoF pózy z jediného RGB obrazu byl dosažen za pomocí konvoluční neuronové sítě, která určí sadu klíčových bodů. Zde bylo využito schopnosti sítě agregovat informace vzhledu objektu přes mnoho úhlů pohledu. Výstupem neuronové sítě je sada heat map, jedna pro každý klíčový bod, kde intenzita značí pravděpodobnost výskytu bodu na dané pozici. Síť měla strukturu přesýpacích hodin, kde obraz v každé vrstvě prošel downsamplingem dokud nebylo dosaženo

nejmenší rozlišení a následným upsamplingem až do původní velikosti. Tento proces umožňuje integraci globálních i lokálních příznaků. V druhém kroku byly klíčové body použity pro úsudek o póze objektu a jeho tvaru v rámci třídy objektu. V případě dostupnosti hloubkových dat lze tento přístup použít k inicializaci algoritmu ICP pro další úpravu pózy. Přístup překonal ve většině případů ty, se kterými byl srovnávan, selhal však v případě objektů, kde došlo k odhalení malého počtu klíčových bodů, což vedlo k jeho nekorektní reprezentaci. Například v případě počítačového monitoru odhalil pouze čtyři takové koplanární body v rozích. Dosažený čas byl 0,3 sekundy pro jeden snímek za použití Intel i7 3.4GHz CPU, 8G RAM a GeForce GTX Titan X 6GB GPU, stinnou stránkou jsou tehdy i hardwarové nároky.

Kouskouridas, Tejani et al. [6] použili základní princip LINEMODu rozšířený o invarianci vůči škále a integrovali jej do nové metody detekce objektů a odhadu pózy nazvanou Hough Forest s latentní distribucí tříd. Původní Hough Forest kombinuje strojové učení s zobecněnou Houghovou transformací a jedná se o sadu rozhodovacích stromů, kde je trénovací proces řízen kombinací randomizace a optimizace. Obrazy jsou rozděleny na menší části (patch) a v každém uzlu stromu mezi nimi probíhá rozdělení množiny do dvou podmnožin. To spočívá v generování mnoha jednoduchých testů porovnávajících jednotlivý pixel, dokud není dosaženo co nejlepšího rozdělení podle testového kritéria. HF s latentní distribucí tříd tento test nahrazuje mírou podobnosti LINEMODu, přičemž invariance vůči škále je dosaženo pomocí hloubkové informace. Rovněž nepotřebuje negativní testovací množinu a v první fázi učení oproti běžnému HF probíhá pouze regrese, kde je měřen zisk informací v rozvětvení každého uzlu. Negativní informace, tedy upřesnění pravděpodobnosti, že po dosažení listu stromu se v obraze skutečně nachází hledaný objekt, se dodatečně získává až dataminingem v druhé, inferenční fázi. Dosažená přesnost sice překonala LINEMOD a jiné aktuální metody, značně ale zaostává v rychlosti detekce přibližně 0,5 FPS, přičemž 1 FPS bylo dosaženo po snížení některých parametrů, což však mělo dopad na přesnost detekce.

Drost [13] navrhl přístup založený na globálním popisu 3D modelu pomocí orientovaných příznaků páru bodů (point pair features) a následném lokálním porovnáváním rychlým hlasovacím schématem, které se podobá zobecněnému Houghovu hlasování a probíhá v redukovaném 2D prostoru. Příznaky jsou asymetrické, popisují relativní pozici a orientaci dvou orientovaných bodů a sestávají se ze čtyř hodnot: vzdálenosti mezi body, dvou úhlů mezi normálou v daném bodu a vektorem definovaného dvěma body a nakonec z úhlu mezi oběma normálami. Přístup dosáhl přesností od 97% v případě menšího zákrytu než 84% a zpracovat jeden obraz trvalo 85 sekund, v případě řídšího vzorkování však dosáhl čtyřicetinásobného zrychlení (0,5 FPS) za cenu snížení přesnosti na 89,2% v případě stejného zakrytí objektu. Je ale nutno uvést, že implementace byla provedena v jazyce Java na 2,00GHz Intel Core Duo T7300 s 1GB paměti za použití jediného vlákna.

V testu na datasetu Hinterstoissera oproti vylepšené verzi LINEMODu[2] Drostův přístup značně zaostával v rychlosti (6,3s na jeden obraz) i přesnosti (79,3%). Není však uvedeno, zda zprostředkovaný algoritmus již byl optimalizovaný, nebo nikoliv.

Brachmann et al. [12] přišli s přístupem, který kombinuje výhody metod založených na lokálních příznacích a z hlediska přesnosti dosahuje na netexturovaných objektech výsledků lepších, než algoritmy postavené na template matchingu, což přináší spoustu konceptuálních a praktických výhod. Například není nutné trénovat oddělené systémy pro texturované a netexturované objekty, pevné a proměnlivé objekty (například nůžky nebo laptop) a objekty v různých stavech (hrnec bez poklice nebo s ní). Dále za použití lokálních příznaků získává robustnost vůči zákrytu.

Algoritmus pro detekci využívá random forest za použití stejných jednoduchých příznaků jako [14], spočívajících v hloubkovém nebo barevném rozdílu pixelů v okolí pixelu i, což zachycuje lokální strukturu objektu. Příznaky jsou navíc invariantní vůči hloubce. Pro trénování byly použity náhodně zvolené pixely ze segmentovaných obrazů objektu a sada RGB-D obrazů zachycujících pozadí. Odhad pózy byl dále určen optimalizací energetické funkce. Na datasetu Hinterstoissera[2] bylo dosaženo 98,3% přesnosti oproti 96,6%, kterých dosáhnul LINEMOD. Dále probělo testování na vlastním datasetu s různým osvícením objektů (umělé jasné světlo - bright, tmavší přirozené světlo - dark a směrové bodové světlo - spot). Trénování proběhlo pouze na prvním dvou typech osvícení a na datasetu bright bylo dosaženo 95% úspěšnosti a 88,2% na datasetu spot. Následné testování LINEMODu na stejném datasetu již mělo podstatně horší výsledky, 80,1% na datasetu bright, což bylo pravděpodobně způsobeno faktem, že objekt je texturovaný. Další testování generalizace osvětlení objektu dosáhlo přesnosti od 55,3% po 70,2%, LINEMOD tedy podstatně hůř detekuje objekty pod různým osvětlením.

Při uvedených výsledcích však Brachmannův přístup trval v průměru 160ms pro random forest a dalších 398ms pro optimalizaci výsledku, tedy méně než 2 FPS. Po snížení parametrů klesl čas optimalizace pod 61ms s přesností 96,4% na datasetu Hinterstoissera[2], což je však stále téměř dvakrát pomalejší. Algoritmus dále prokázal dobrou škálovatelnost vůči počtu objektů a možných póz s menším než lineárním růstem složitosti výpočtu.

Liebelt a Schmid [21] využili kombinace modelu 2D vzhledu s externí 3D geometrií za použití deskriptoru DAISY [22]. Vzhled třídy objektu je naučen z databáze 2D obrazů, které zobrazují daný objekt z různých bodů pohledu. Ty jsou rovněž označeny anotací jejich pózy a 2D bounding boxem, který je dále rozdělen do pravidelné mřížky, kde každý její blok reprezentuje část objektu. Pro detekcí celých oblastí zájmu v obraze je použit jediný pyramidový detektor, zatímco pro každou oblast částí pro každý bod pohledu je použito několik menších, překrývajících se detektorů. 3D geometrie je naučena z jednoho nebo několika 3D CAD modelů reprezentujících geometrii dané třídy objektu. Modely jsou renderovány z mnoha bodů pohledu a obrazy dále rozděleny do pravidelné mřížky stejně jako v případě předchozím. Pro každý renderovaný pixel je známa jeho původní pozice na CAD modelu, pixely obrazu patřící do stejné části objektu mohou být tedy zpětně projektovány na povrch a samplovány do diskrétních 3D bodů. Distribuci všech 3D bodů patřící jedné části objektu pak lze modelovat kombinací gaussiánů. Výsledná reprezentace třídy objektu sestává z 2D pre-detektoru oblastí zájmu, hustého 2D detektoru částí pro každý bod pohledu a přibližné reprezentace 3D geometrie třídy objektu. Výsledky testování na 3D Object Category datasetech CAR a BICYCLE však dosáhly pouze přesností $76,7\%$ a

69, 8%, což již novější metody snadno překonaly.

Tombari, Salti a Di Stefano [28] se zaměřili na 3D deskriptory, které rozdělili do dvou kategorií. První z nich jsou signaturové, které stály na počátku této oblasti a popisují 3D povrch v blízkosti daného bodu (support) definováním lokálního a invariantního referenčního rámce a enkódováním geometrických vlastností spočítaných na každém bodu podmnožiny supportu. Oproti tomu histogramové metody popisují support akumulací lokálních geometrických nebo topologických vlastností (např. počty bodů, velikost plochy trojúhelníků daného modelu) do histogramů podle kvantizované domény (např. souřadnice bodů nebo zakřivení), což požaduje definici referenční osy nebo lokálního referenčního rámce. Signaturové metody mají vysoký deskriptivní potenciál díky použití dobře prostorově lokalizovaných informací, oproti tomu histogramy nabízejí vyšší robustnost na úkor deskriptivní síly. Dále se zaměřili na důležitost a vlastnosti referenčního rámce, který by měl být unikátní a jednoznačný, z nichž zatím dostupné metody splňovaly pouze jedno nebo druhé. Na základě těchhle pozorování navrhli nový deskriptor, který byl inspirován 2D deskriptorem SIFT [30] pomocí zaměření se na vlastnosti, díky kterým byl SIFT jedním z nejúspěšnějších a nejpoužívanějších. První z těchto vlastností je používání histogramů skrze celým algoritmem, od definice lokální orientace po samotný deskriptor, což má za důsledek jeho robustnost. Dále jsou prvky lokálních histogramů založené na derivacích prvního řádu popisujících zajímavý signál, což je zde gradient intenzit. I přes argumenty, že deskriptor založený na diferenciálních entitách může vést k špatné robustnosti vůči šumu, poskytuje vysokou deskriptivní sílu. Výsledkem byl deskriptor Signature of Histograms of OrienTations (SHOT) založený na signatuře histogramů, tedy kombinací obou známých přístupů. Ten enkóduje histogramy základních diferenciálních entit prvního řádu (normály bodů uvnitř supportu) které reprezentují lokální strukturu povrchu lépe než obyčejné 3D souřadnice. Použití histogramů s sebou přináší filtrující efekt, kterým je dosažena robustnost vůči šumu. S definicí unikátního a robustního 3D referenčního rámce je rovněž možné zvýšit diskriminační sílu deskriptoru zavedením geometrické informace obsahující lokaci bodů uvnitř supportu, což napodobuje signaturové metody.

V další práci [29] jej rozšířili o texturované objekty v deskriptoru Color-SHOT (CSHOT) přidáním druhé signatury (histogramu barvy), skládající se z tříprvkového vektoru barevných intenzit. CSHOT má stejné parametry jako SHOT (poloměr supportu a počet přihrádek histogramu), vzhledem k odlišné povaze signatur je však užitečné povolit odlišný počet přihrádek v obou typech histogramů. Získává tedy další parametr, což je počet přihrádek v histogramu textury a nazývá se Color Step. V porovnání s původním deskriptorem byl efektivnější především při použití barevných reprezentací RGB a CIELab, došlo však ke zpomalení o polovinu.

Li et al. [31] se zaměřili především na detekci ve scénách s velkým množstvím objektů. Zde je několik hlavních problémů. Prvním z nich je rozpoznání objektů bez omezení jejich geometrických a povrchových vlastností. Různé metody detekce objektů toto řeší pomocí invariantních lokálních deskriptorů jako jsou výše uvedené SIFT a CSHOT. Jejich výkon však značně klesá v případě netexturovaných objektů, scén s velkým množstvím objektů a velkých rozdílů v bodech pohledu mezi tréninkovými a testovacími daty. Metody template matchingu mají podobné pro-

blémy, navíc jsou špatně škálovatelné s rostoucím počtem detekovaných objektů a málo robustní vůči zákrytu. Dalším problémem je rozlišení mezi různými typy objektů s podobným tvarem, např. různé druhy vrtaček na obrázku . Většina metod používající globální, ručně vytvořené příznaky jako například LINEMOD není schopna zachytit vizuální detaily a současně zachovat invariantní reprezentaci objektu. Značný pokrok v problematice přesného rozpoznání instancí objektu byl docílen pomocí konvolučních architektur [32], jejich přizpůsobení pro detekci pózy je však netriviální, jelikož byly optimalizovány pouze pro klasifikaci. Pro vyřešení problému navrhli nový přístup sestávající z filtrování 3D dat pomocí příznaku CSHOT, generalizované sdružovací (ang. pooling) metody na konstruování sdružených příznaků ze dvou domén - SIFT pro gradienty a FPFH [33] pro 3D geometrii a dvouúrovňové sémantické segmentace pomocí SVM. Přesností překonali dostupné metody, rychlost však dosahovala pouze 0,2 FPS na desktopu s Intel Xeon CPU E5-2690. Nejpomalejší části algoritmu (výpočet CSHOT a FPFH) však lze značně urychlit pomocí výpočtů na GPU.

2.3 Částicový a Kalmanův filtr

Obě tyto metody mají využití v sledování pohybu objektu v čase a fungují na principu rekurzivního aktualizování odhadu stavu na základě předchozích pozorování. Kalmanův filtr toho dosahuje lineární projekcí a pracuje s lineární reprezentací zatíženou šumem, který má tvar gaussiánu, zatímco částicový filtr využívá sekvenční metodu Monte Carlo a pracuje v prostředí, které může být nelineární nebo mít tvar jiný než gaussián, v těch případech sice dosáhne stejného výsledku jako Kalmanův filtr, ale v delším výpočetním čase [7].

Changhyun a Henrik [8] použili na sledování pózy objektu masivně paralelizovaný částicový filtr na GPU spolu s 3D modelem sledovaného objektu. V použitých příznacích byly zahrnuty barvy, normály a 3D pozice částic a při použití několika tisíců částic dosáhli rychlosti kolem 20 FPS, přičemž rychlost výpočtu rostla s počtem částic lineárně.

2.4 Event-based vision

Tento přístup využívá speciální hardware, tzv. event-based kameru, příkladem je kamera Dynamic Vision Sensor (DVS). Zatímco klasická kamera dává pevný počet výstupů v pevných intervalech (např. 60 FPS), výstup event-based kamery je asynchronní a generován s přesností na mikrosekundy pokaždé, když změna hodnoty pixelu překročí danou prahovou hodnotu. Výhodou těchto zařízení je vysoký dynamický rozsah (120 dB), nízká latence, malá šířka přenosového pásma a nízká spotřeba energie, mají však velmi malé rozlišení (128×128 pixelů u DVS) a vyžadují na rozdíl od klasických kamer zcela nové algoritmy [10].

3 Teoretická část

V následující kapitole budou více přiblíženy metody zvolené pro realizaci cíle práce.

3.1 Volba metod

Postup práce jsem zvolil na základě následujících požadavků a cílů:

- Cílem je určení 6 DoF pózy jednoho zvoleného netexturovaného objektu
- Vstupní data jsou ve formátu RGB-D
- Jsou dostupné 3D modely detekovaných objektů
- Detekce musí být provedena v reálném čase, jako dolní hranici jsme stanovili 1 FPS
- Algoritmus musí být odolný vůči zákrytům

Po uvážení požadavků a analýze výhod a nevýhod již dostupných metod jsem se rozhodl pro použití postupu založeném na metodě LINEMOD [2], jelikož splňuje naše požadavky včetně časových i na běžném hardwaru a její hlavní nevýhoda (lineární růst složitosti s počtem objektů, přičemž dvojnásobný počet objektů vede k zpomalení na polovinu) není v rozporu se zadáním.

3.2 LINEMOD

Následující podkapitola je zpracována podle původního článku metody [1]. Jak již bylo zmíněno v předchozí kapitole, LINEMOD je metoda založená na template matchingu, kde pravděpodobnost výskytu objektu určuje míra podobnosti mezi vstupním a vzorovým obrazem shodné velikosti. Ta je určená následujícím vzorcem:

$$\mathcal{E}(I,T,c) = \sum_{r \in P} \left(\max_{t \in R(c+r)} |\cos\left(ori\left(O,r\right) - ori\left(I,t\right)\right)| \right) \tag{1}$$

kde $ori\left(O,r\right)$ je orientace gradientu na pozici r ve vzorovém obraze O a P je seznam bodů, se kterými se z daného vzorového obrazu pro bod c v obraze vstupním počítá. Vzorový obraz T je tedy definován jako pár $T = (O,P)$.

Stejně tak $ori\left(I,t\right)$ je orientace gradientu ve vstupním obraze I, přičemž $t \in R\left(c+r\right)$, kde r je posun vzhledem k bodu c ve vstupním obraze, pro který podobnost počítáme. Dále

$$R\left(c+r\right) = \left[c+r-\frac{N}{2}, c+r+\frac{N}{2}\right] \times \left[c+r-\frac{N}{2}, c+r+\frac{N}{2}\right] \tag{2}$$

definuje okolí velikosti N se středem v bodě $c+r$ ve vstupním obraze.

Pro každou orientaci gradientu vzorového objektu je tedy ve vstupním obraze vyhledáván co nejvíce podobný gradient, který může být od prohledáváného bodu vzdálený až o $\frac{T}{2}$ pixelů. Tento posun činí detektor robustní vůči drobným posunům a deformacím.

3.2.1 Výpočet gradientů

Algoritmus využívá orientace gradientů kvůli jejich robustnosti vůči šumu a osvětlení, navíc se prokázalo, že lépe rozlišují různé části obrazu oproti jiným metodám [19, 30]. Navíc jsou často jedinou spolehlivou vlastností obrazu pro detekci netexturovaných objektů. Použití pouze orientace gradientů a ne jejich velikosti rovněž míru podobnosti činí robustní vůči změnám kontrastu. Použití absolutní hodnoty kosinu navíc umožňuje korektní zpracování hranic, jelikož výsledek nebude ovlivněn světlým nebo tmavým pozadím.

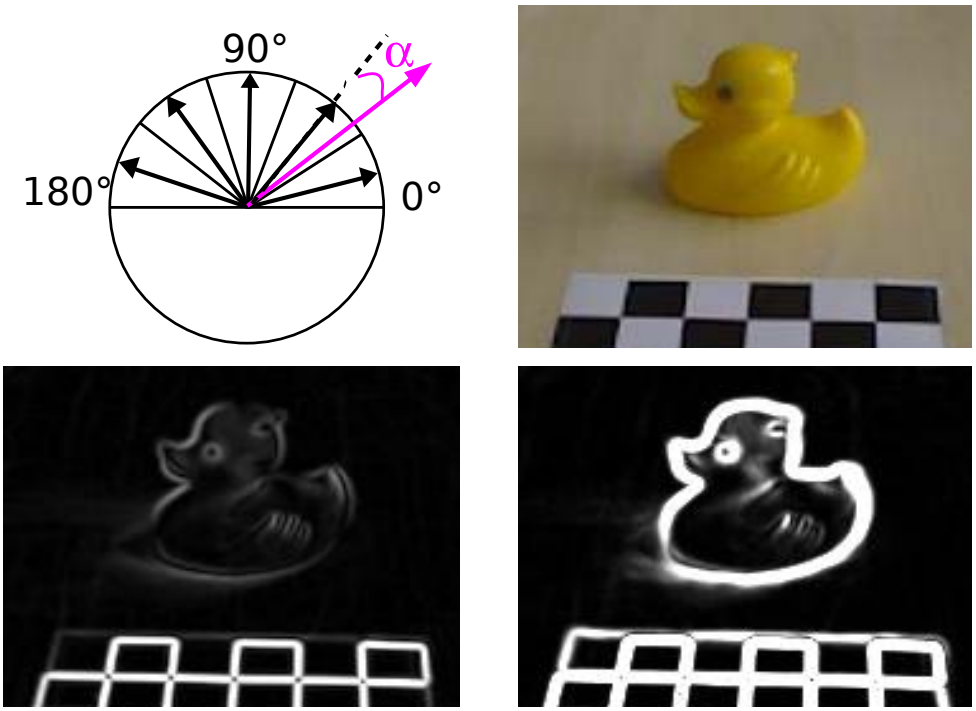
Robustnost dále zvyšuje výpočet orientace gradientu odděleně na každém barevném kanálu obrazu a následné použití toho kanálu, na kterém se nachází gradient s největší velikostí. Na RGB obrazu I tedy vypočteme orientaci gradientu $I_g(x)$ v bodě x pomocí

$$I_g(x) = ori(\hat{C}(x)) \tag{3}$$

kde

$$\hat{C}(x) = \underset{C \in \{R,G,B\}}{\operatorname{argmax}} \left\| \frac{\partial C}{\partial x} \right\| \tag{4}$$

a R, G a B jsou barevné kanály obrazu.



Obrázek 2: **Vlevo nahoře:** Kvantizace orientací gradientů, gradient růžové barvy je nejblíže druhé příhrádce. **Vpravo nahoře:** Zdrojový obraz s modelem. **Vlevo dole:** Obraz gradientů spočítán na greyscale obraze. **Vpravo dole:** Gradienty spočítány novou metodou, jsou zde znatelně viditelnější. Zpracováno podle[1].

Pro kvantizaci orientace gradientů je vypuštěn i jejich směr a prostor orientací je rovnoměrně rozdělen do n_0 rozsahů. Každému bodu se poté přiřadí ten gradient, jehož kvantizovaná orientace se nejčastěji vyskytuje v okolí velikosti 3×3, což je učiní robustní vůči šumu. Dále jsou zachovány pouze ty gradienty, jejichž velikost je větší než nějaká malá prahová hodnota. Neoptimizovaný proces pro VGA obraz trvá na CPU přibližně 31ms.

3.2.2 Výpočet response mapy

Opakovanému vyhodnocování operátoru max pro každé porovnávání vzorového obrazu oproti vstupnímu se lze vyhnout pomocí binární reprezentace gradientů v blízkosti místa v obraze, která je značena J. Rychlého zpracování je poté docíleno použitím této reprezentace spolu se strukturou lookup table pro předvypočítání maximálních hodnot.

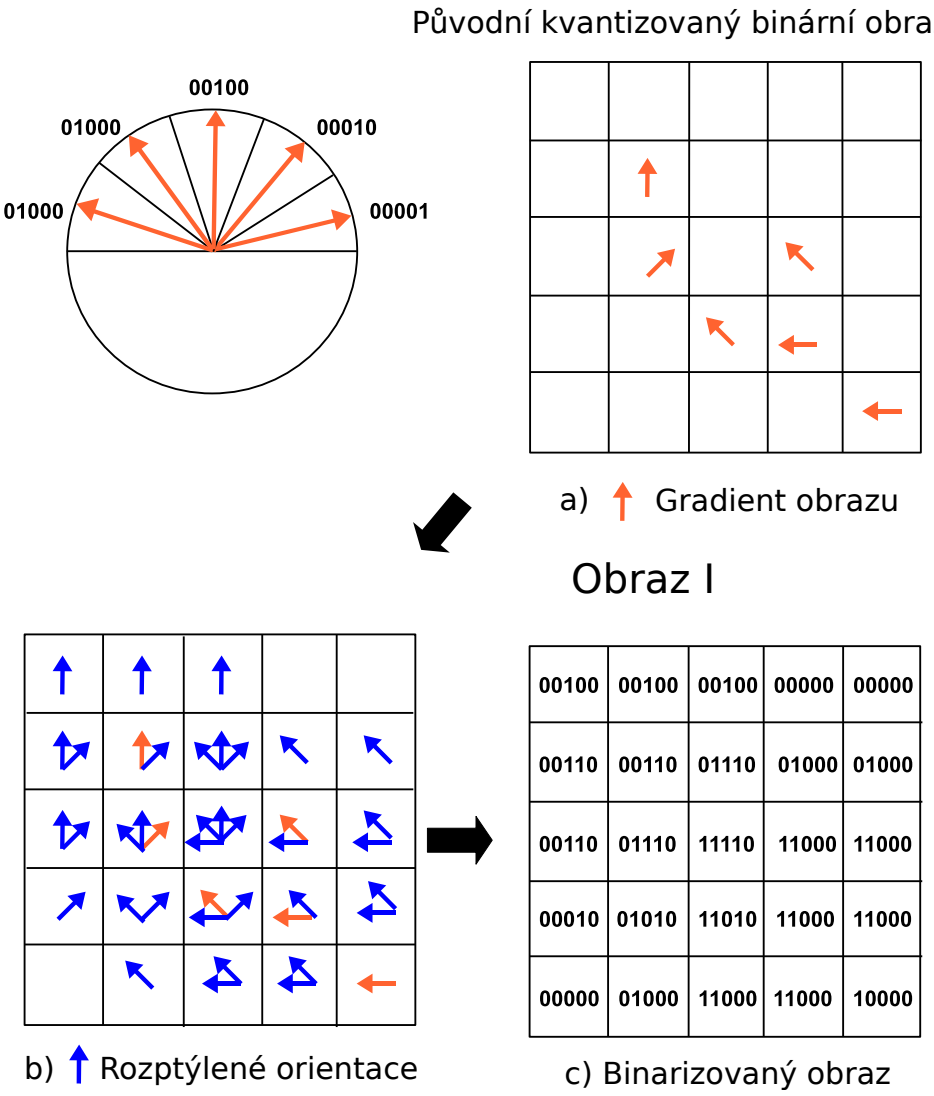
Nejprve je provedena kvantizace orientací do již výše uvedených n_0 hodnot, což umožňuje rozprostření orientací $ori\,(I,t)$ kolem jejich pozice pro získání nové reprezentace vstupního obrazu. Pro rychlost jsou možné kombinace orientací rozprostřených do bodu obrazu m zakódovány jako binární řetězec, přičemž každý bit koresponduje s jednou kvantizovanou orientací a je nastavený na 1, pokud se daná orientace gradientu vyskytuje v okolí bodu m. Tyto řetězce tvoří obraz J a jsou použity jako index pro přístup do struktury lookup table pro rychlé předpočítání míry podobnosti. J lze vypočítat velmi rychle: Nejprve proběhne výpočet mapy pro každou kvantizovanou orientaci, jejichž hodnoty jsou nastaveny na 1 nebo 0 podle výskytu nebo absence dané orientace. Výsledný obraz J je poté získán pouhým posouváním map v rozsahu $\left[-\frac{N}{2}, +\frac{N}{2}\right] \times \left[-\frac{N}{2}, +\frac{N}{2}\right]$ a sloučením těchto posunutých map bitovou operací OR.

Jak již bylo uvedeno, obraz J je použitý spolu se strukturou lookup table pro předpočítání hodnoty operace max pro každý bod a každou možnou orientaci $ori\,(O,r)$ ve vzorovém obraze. Výsledky jsou uloženy ve 2D mapách S_i. Evaluace míry podobnosti je poté provedena pouhým součtem hodnot přečtených z S_i. Lookup table τ_i pro každou z n_0 kvantizovaných orientací je vypočten

$$\tau_i\,[L] = \max_{l \in L} \|\cos\,(i-l)\|\,, \tag{5}$$

kde

- i je index kvantizované orientace, který je pro jednoduchost značení použit i pro reprezentaci korespondujícího úhlu v radiánech
- L je seznam orientací v okolí gradientu s orientací i. Číselná hodnota korespondující s binární reprezentací L je použita jako index prvku v lookup table.



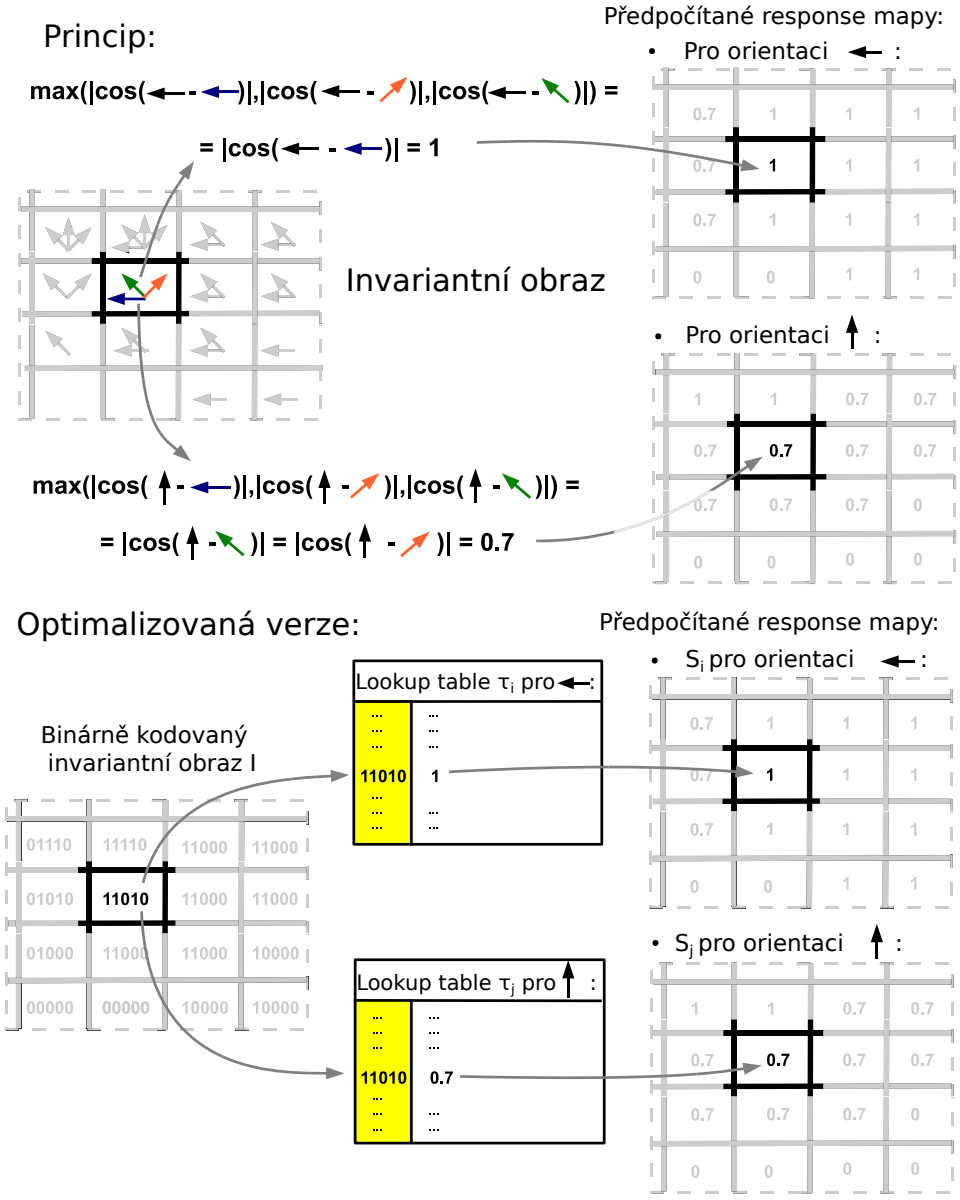
Obrázek 3: Rozprostření orientací gradientů. **Vlevo nahoře**: Orientace gradientů a jejich binární kódy, směr není brán v potaz. **a)** Orientace gradientů ve vstupu jsou získány a kvantizovány. **b)** Místa okolo každé orientace jsou také označena touto orientací, značeno modrými šipkami. **c)** Účinná reprezentace orientací po předchozí operaci. Zpracováno podle[1].

Pro každou orientaci i lze pak vypočítat hodnotu v každém bodě c response mapy S_i jako

$$S_i\left(c\right) = \tau_i\left[J\left(c\right)\right]. \tag{6}$$

Míra podobnosti je potom:

$$\mathcal{E}(I,T,c) = \sum_{r\in P} S_{ori(O,r)}\left(c+r\right). \tag{7}$$



Obrázek 4: Předpočítání response mapy S_i. **Nahoře**: Každá kvantizovaná orientace má jednu response mapu ukládající maximální podobnost mezi korespondující orientací a orientacem v invariantním obraze. **Dole**: Rychlý výpočet pomocí binární reprezentace seznamu orientací, který slouží jako lookup table pro maximální podobnost. Zpracováno podle[1].

3.2.3 Linearizace paměti pro paralelizaci

Pomocí vzorce 7 lze porovnat vzor oproti celému vstupnímu oprazu pouhým sečtením hodnot v response mapách S_i. Jedna z výhod rozprostření orientací je však ta, že je dostačující evaluovat pouze každý T-tý pixel bez redukce schopnosti rozpoznávat. Pro účinné využití této vlastnosti je nutno vzít v potaz architekturu moderních počítačů.

Moderní procesory nečtou hodnoty dat z hlavní paměti jednu po jedné, ale několik současně, tzv. *cache line*. Přístup do paměti v nahodilých místech způsobuje *cache miss* a zpomaluje výpočty. Naopak načítání několik hodnot ze stejné cache line je velmi rychlé. Důsledkem je, že ukládání dat ve stejném pořadí, v jakém jsou načítána, značně zrychlí výpočty. Navíc to umožňuje paralelizaci: Například pokud používáme 8-bitové hodnoty jako v případě našich S_i map, SSE instrukce dokáží provést operace paralelně na 16 hodnot. Na vícejádrových procesorech nebo GPU lze provést ještě více současných operací.

Předpočítané response mapy S_i se tedy uloží do paměti v přívětivější podobě pro cache: Každá z nich se přestrukturuje tak, že hodnoty jednoho řádku, co jsou od sebe T pixelů na ose x jsou v paměti vedle sebe. Po zpracování řádku se pokračuje řádkem vzdáleným o T pixelů na ose y. Výpočet míry podobnosti pro obraz pak lze provést pouhým sečtením linearizovaných pamětí s příslušným offsetem vypočteným z pozic r ve vzorech.

3.2.4 Rozšíření pro hloubkové senzory

V případě dostupnosti hloubkové mapy je možné robustnost algoritmu dále zvýšit. Podobně jako v případě barevných kanálů, hloubkový obraz používá kvantizované normálové vektory povrchu vypočítané z hloubkové mapy.

Ty lze rychle vypočítat následujícím způsobem, přičemž pro každý pixel na pozici x uvažujeme Taylorův rozvoj prvního řádu hloubkové funkce $D(x)$:

$$D(x+dx) - D(x) = dx^{\top} \bigtriangledown D + h.o.t. \tag{8}$$

Pro oblast definovanou kolem x, každý pixelový offset dx dává rovnici, která omezuje hodnotu ΔD, což nám dovoluje odhadnout optimální gradient ΔD nejmenším čtvercem. Tenhle hloubkový gradient koresponduje s 3D plochou, procházející body X, X_1, X_2:

$$X = \overrightarrow{v}(x)\, D(x), \tag{9}$$

$$X_1 = \overrightarrow{v}\left(x + [1,0]^{\top}\right) D\left(x + [1,0] \bigtriangledown D\right), \tag{10}$$

$$X_2 = \overrightarrow{v}\left(x + [0,1]^{\top}\right) D\left(x + [0,1] \bigtriangledown D\right), \tag{11}$$

kde $\overrightarrow{v}(x)$ je vektor směru pohledu procházející pixelem x a je vypočítán z parametrů hloubkového senzoru. Normálový vektor povrchu který se promítne na x lze odhadnout jako normalizovaný vektorový součin $X_1 - X$ a $X_2 - X$. Tahle metoda by však nebyla robustní v okolí obrysů zákrytu, kde aproximace prvního řádu 8 neplatí. Podobně jako při bilaterální filtraci jsou ignorovány pixely, jejichž hloubkový rozdíl oproti středovému pixelu přesahuje určitou prahovou hodnotu. To znamená, že tento přístup vyhladí kvantizační šum na povrchu a současně stále poskytuje smysluplné odhady normál povrchu v blízkosti silných hloubkových diskontinuit.

Míra podobnosti je poté definována jako skalární součin normalizovaných normál povrchu, zbytek funguje stejně jako v případě gradientů pro RGB obraz. Celková míra podobnosti se rovná součtu podobností pro gradienty a normály.

3.2.5 Preprocessing

LINEMOD pracuje se dvěma příznaky: gradienty barevného obrazu a povrchovými normálami, přičemž oba jsou redukovány diskretizací na nízký počet hodnot. Vzory jsou z nich vypočítány hustým přístupem, lze však pracovat jen z podmnožinou těchto příznaků, což dále zrychlí detekci bez ztráty její přesnosti.

V případě existence 3D modelu hledaného objektu jej lze využít pro automatizované vytvoření vzorových obrazů, což poskytuje mnoho výhod oproti učení online. Zaprvé není potřeba lidského úsilí, navíc je obvykle zapotřebí osoby znalé v dané problematice pro nasnímání objektu v celém rozsahu póz s vhodným samplingem. Online metody rovněž obvykle používají greedy přístup a optimální poměr výkonu a robustnosti není zaručen. Navíc 3D model umožňuje snadné získání dalších informací, např. přesná maska obrazu tělesa, čehož lze dále využít k vytvoření bounding boxů pro jednotlivé templaty.

Z barevných gradientů se zachovají pouze ty, které se nacházejí na obrysu siluety tělesa z důvodu zaměření na netexturované objekty, které uvnitř této siluety nemají žádné nebo velmi málo příznaků. Textura daného CAD 3D modelu rovněž není vždy dostupná.

Siluetu lze získat projekcí 3D modelu v dané póze a její obrys pouhým odečtením erodované siluety od té původní. Poté proběhne spočítání všech gradientů na obryse a jejich seřazení podle velikosti, což je důležité, jelikož nelze zaručit, že obrys má šířku jeden pixel. Seřazeným seznamem obrysů se následně iteruje greedy přístupem od největší velikosti gradientu a ze seznamu se odstraní gradienty, jejichž vzdálenost od vybraného prvku je dostatečně malá podle určité prahové hodnoty. Pokud iterace skončí před výběrem požadovaného počtu příznaků, sníží se práh vzdálenosti o 1 a proces se opakuje, přičemž práh je nastaven na poměr oblasti pokryté siluetou a žádáného počtu příznaků.

Opačný případ nastane u povrchových normál, kde jsou považovány pouze příznaky uvnitř siluety objektu. Důvodem je skutečnost, že normály na okraji objektu jsou často neodhadnuty přesně, nebo nebyly získány vůbec, přičemž ty, které jsou obklopeny normálami stejné orientace jsou spolehlivěji získány. Při tvorbě vzorových obrazů jsou tyhle normály ponechány a ty méně stabilní zahozeny. Toho je docíleno tvorbou masky pro každou z osmi možných orientací diskretizovaných normál objektu v dané póze. Pro každou masku je poté vážena každá normála s vzdáleností od hranice masky, přičemž větší vzdálenosti indikují normálu obklopenou normálami podobné orientace a malé vzdálenosti naopak normály obklopené jinými normálami, ne však v každém případě z důvodu nepřesnosti kolem hranic objektu.

Nejprve jsou tedy zamítnuty normály s váhou menší, než nějaká určitá vzdálenost, v praxi se používá hodnota 2. Nelze však spoléhat při zachování normál pouze na jejich váhy, jelikož velké oblast s podobnými normálami by na výsledný obraz měly příliš velký vliv, nejprve se tedy

normalizují váhy podle velikosti masky příslušného objektu. Postup je poté stejný jako u gradientů barvy, vytvoření seznamu normál seřazených podle jejich váhy a výběr konečných příznaků iterativním způsobem. Tenhle postup zaručuje kvaziuniformní rozptyl vybraných normál.

3.2.6 Postprocessing

Každý vzorový obraz detekovaný LINEMODem poskytuje hrubý odhad pózy objektu, který je dostatečně přesný pro rychlou kontrolu barevné informace. V potaz se berou pixely, které leží na projekci objektu v odhadované póze a kontrola probíhá na základě počtu pixelů s odpovídající barvou. Porovnává se pouze odstín (hue) detekovaných pixelů s odstínem objektu a pokud jejich rozdíl (modulo 2π) je menší než prahová hodnota, přijímá se jako pixel objektu. Porovnávání pouze na základě odstínu činí test robustní vůči změnám osvětlení. Pokud procento pixelů s očekávanou barvou není dostatečně velké (v případě [2] 70%), detekce se považuje jako false positive. V praxi se nepovažují pixely, které jsou příliš blízko hranice projekce objektu kvůli toleranci vůči nepřesnosti hrubého odhadu pózy. To lze rychle provést erozí projekce (masky) objektu.

Výše uvedená metoda však nefunguje pro černé a bílé objekty, jelikož nejsou pokryty složkou odstínu. Tenhle problém lze vyřešit jejich namapováním na podobné odstíny: černé objekty na modré a bílé na žluté. Takové objekty jsou odhaleny kontrolou jejich sytosti a jasu. V případě, že jasová složka je menší než práh t_v, odstín se nastaví na modrou barvu. Pokud je jasová složka větší než práh t_v a sytost větší než prahová hodnota t_s, nastaví se odstín na žlutou. V případě [2] $t_s = t_v = 0,12$.

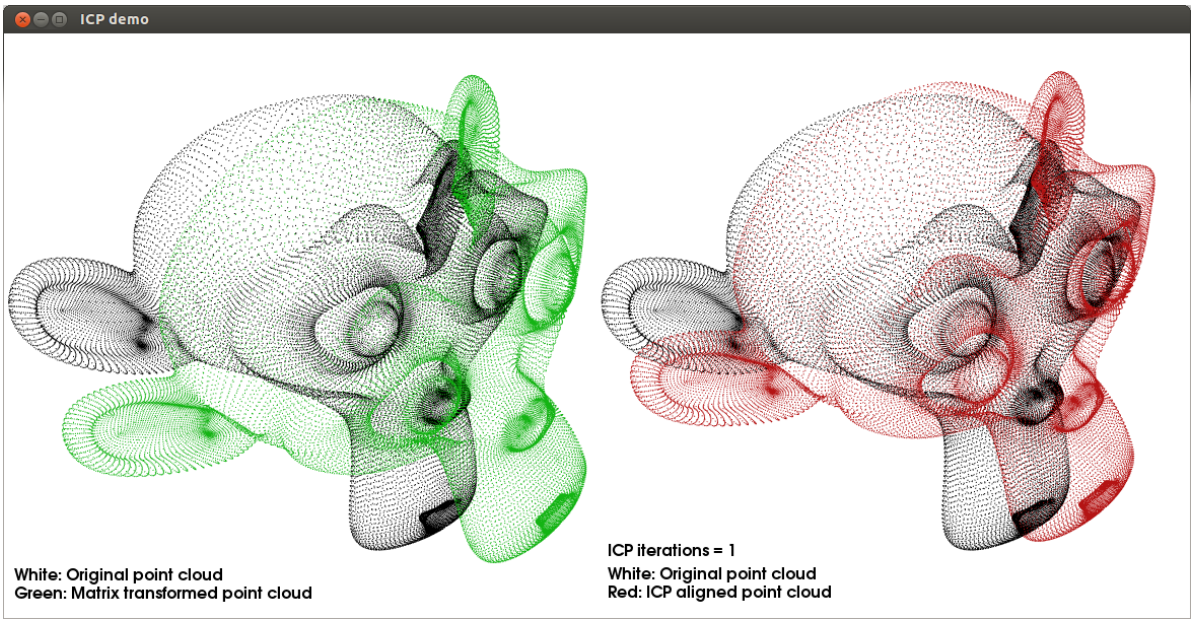
Pro detekce, které prošly kontrolou barvy, proběhne následně upřesnění hrubého odhadu pózy algoritmem Iterative Closest Point (ICP), který minimalizuje vzdálenost mezi dvěma sadami bodů, v našem případě mezi 3D body z hloubkové mapy detekce a 3D body modelu objektu. Pro rychlejší zpracování lze nejprve redukuje počet bodů subsamplingem, např. voxelovým filtrem. Pro robustnost v každé iteraci i probíhá výpočet transformace pouze pomocí vnitřních (inlier) bodů, což jsou ty body, jejichž vzdálenost od 3D modelu je menší, než adaptivní prahová hodnota t_i. t_0 je inicializována na velikost objektu, , t_{i+1} na trojnásobek průměrné vzdálenosti těchto bodů od modelu v čase i. Pokud je i po konvergování průměrná vzdálenost od modelu příliš velká, detekce se zamítá jako false positive. Tento proces se opakuje, dokud jím neprojde $n = 3$ detekcí. Poté je proveden pomalejší a přesnější ICP pro těchto n detekcí za použití všech bodů na projekci objektu nebo v její blízkosti, přičemž nejlepší odhad je ten, jehož průměrná vzdálenost mezi vnitřními body a 3D modelem je nejmenší. Posledním krokem je hloubkový test, který spočívá ve spočítání pixelů ležících na projekci objektu a porovnání jejich hloubky oproti očekáváné hodnotě. Přijímají se ty pixely, jejichž hloubkový rozdíl nepřekračuje určitou prahovou hodnotu a detekce je přijata, pokud alespoň 70% pixelů má očekávanou hloubku, jinak proběhne zamítnutí detekce jako false positive.

3.3 Iterative Closest Point

Úkolem algoritmu ICP je minimalizovat vzdálenost mezi dvěma množinami bodů, v případě výstupu algoritmu LINEMOD mezi 3D body vzorového obrazu s hrubým odhadem pózy a 3D body nalezenými ve vstupním obrazu, které obsahují hledaný objekt ve skutečné póze. Finální odhad 6 DoF pózy je tedy získán určením rotace a translace bodů vzorového objektu tak, aby jejich vzdálenost od bodů reálného obrazu byla co nejmenší. Podmínkou pro úspěšné nalezení finální transformace je dostatečně podobný prvotní odhad, jinak může dojít k uváznutí vzdálenosti mezi množinami bodů v lokálním minimu.

Tento algoritmus lze rozdělit do několika úloh:

- výběr podmnožiny bodů, které budou pro výpočet použity
- nalezení korespondujících bodů mezi množinami vstupních bodů
- vážení nalezených korespondencí a odstranění nevhodných párů bodů
- určení transformace mezi korespondujícími body, což lze provést jednotkovým rozkladem matice (SVD)

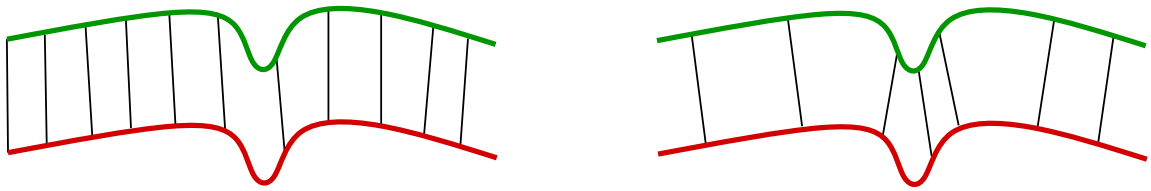


Obrázek 5: Ilustrace algoritmu ICP. Cílem je minimalizovat vzdálenost mezi množinami bodů, v každé iteraci se vzdálenost zmenšuje, dokud není dosaženo podmínky ukončení. Zdroj: http://pointclouds.org/documentation/tutorials/_images/icp-1.png

Algoritmus probíhá rekurzivně, dokud není dosažena podmínka pro zastavení, např. dosažení vzdálenosti menší než zadaná prahová hodnota nebo provedení určeného počtu iterací.[25]

3.3.1 Výběr podmnožiny bodů

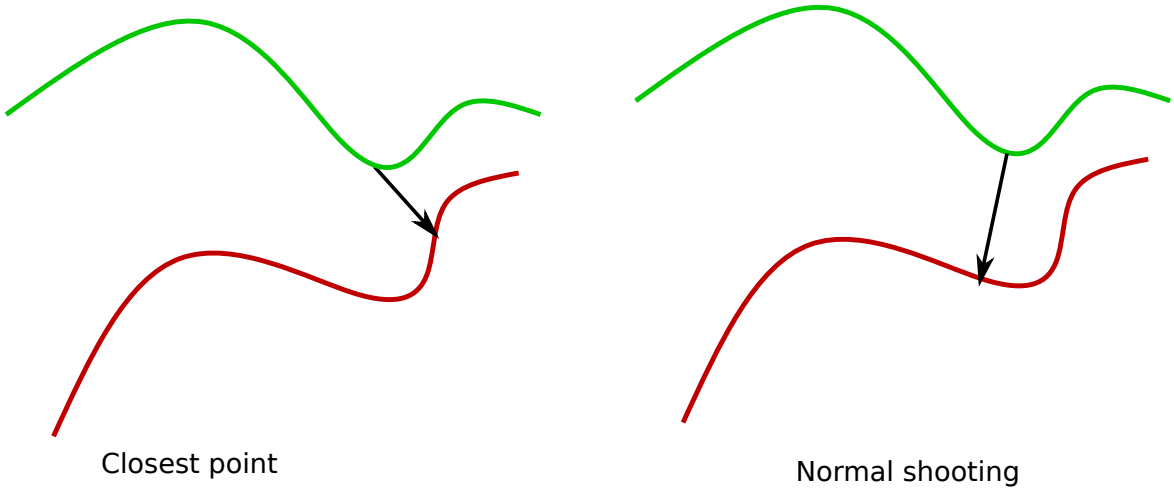
Cílem tohoto kroku je zvolení podmnožiny bodů dále použitých pro výpočet transformace tak, aby vybraná podmnožina bodů dobře reprezentovala hledaný objekt a aby současně došlo k co největší redukci výpočetního času. Nejjednodušší možností je krok zcela přeskočit a použít všechny dostupné body, což však není praktické u velkých objektů. Dalšími jednoduchými možnostmi je použít sampling uniformní, kde ovšem může nastat ztráta jemných příznaků (např. drobné záhyby) nebo sampling náhodný, který je pro obecné objekty o něco vhodnější. Další možností je použít normal sampling, kde se body vkládají do jednotlivých přihrádek podle jejich normál a následně z přihrádek uniformně vybírají. Tahle metoda je robustní hlavně pro převážně hladké povrchy s řídkými příznaky, jako např. plocha se zářezy.[25]



Obrázek 6: Příklad uniformního výběru bodů a na základě normal samplingu. Zpracováno podle[25].

3.3.2 Nalezení korespondujících bodů

Vhodné nalezení korespondencí je ta část, která má největší podíl na rychlosti chodu algoritmu a na tom, jak dobře bude konvergovat. Nejprostším způsobem je nalezení nejbližšího bodu v druhé množině, což je obvykle robustní a stabilní řešení pro tělesa se složitými geometrickými tvary, určení korespondencí je však pomalé a vyžaduje preprocessing. O něco rychlejším řešením pro hladké povrchy je normal shooting, kde je z bodu ve směru jeho normály projektován paprsek podobně jak v ray tracingu a zvolen ten bod, který protne, což lze urychlit pomocí BSP nebo 3D stromů. U zašumělých množin bodů nebo komplexních těles je však méně vhodným řešením. Předchozí varianty lze urychlit přístupem Closest Compatible Point, který páruje pouze s kompatibilními body např. na základě jejich normál nebo barev. V krajním případě metoda směřuje k feature matchingu. [25]



Obrázek 7: Přístupy výběru korespondujících bodů closest point a normal shooting. Zpracováno podle[25].

Nalezeným párům bodů se následně přiřadí váha podle vybraného kritéria. Některé běžně používané možnosti jsou: [26]

- Konstantní váha pro všechny páry bodů.
- Přiřazení nižšíh vrah párům, které jsou dále od sebe. Tenhle přístup je podobný dalšímu kroku, což je úplné odstraňování párů s větší vzájemnou vzdáleností, než je nějaká prahová hodnota. Na rozdíl od úplného odstranění však nedojde k vzniku diskontinuit. Lze např. použít

$$Weight = 1 - \frac{Dist\left(p_1, p_2\right)}{Dist_{max}}. \tag{12}$$

- Vážení na základě kompatibility barev bodů nebo normál, kde

$$Weight = n_1 \cdot n_2. \tag{13}$$

3.3.3 Nalezení finální transformace

Vstupem algoritmu jsou dvě množiny 3D bodů $X = \{x_1, ..., x_n\}$ a $P = \{p_1, ..., p_n\}$ a cílem je získání translace t a rotace R tak, aby došlo k minimalizaci součtu čtverců odchylek $E\left(R, t\right)$, přičemž

$$E\left(R, t\right) = \frac{1}{N_p} \sum_{i=1}^{N_p} \left\| x_i - Rp_i - t \right\|^2, \tag{14}$$

kde x_i a p_i jsou korespondující body. Prvním krokem před vypočtením transformace je odečtení korespondujícího těžiště μ od každého bodu v obou množinách. Těžiště se vypočítají pomocí

$$\mu_x = \frac{1}{N_x}\sum_{i=1}^{N_x} x_i, \mu_p = \frac{1}{N_p}\sum_{i=1}^{N_p} p_i, \tag{15}$$

výsledné množiny bodů pak jsou

$$X' = \{x_i - \mu_x\} = \{x_i'\}, P' = \{p_i - \mu_p\} = \{p_i'\}. \tag{16}$$

.

Rovnici 14 poté lze přepsat jako

$$E(R,t) = \frac{1}{N_p}\sum_{i=1}^{N_p} \left\| x_i' - Rp_i' - \underbrace{(t - \mu_x + R\mu_p)}_{\tilde{t}} \right\|^2 \tag{17}$$

a po umocnění

$$E(R,t) = \frac{1}{N_p}\sum_{i=1}^{N_p} \left\|x_i' - Rp_i'\right\|^2 - \frac{2}{N_p}\tilde{t}\cdot\sum_{i=1}^{N_p}(x_i' - Rp_i') + \frac{1}{N_p}\sum_{i=1}^{N_p}\left\|\tilde{t}\right\|^2. \tag{18}$$

Jelikož $E(R,t)$ je potřeba minimalizovat a druhý a třetí term je nulový v případě, že $\tilde{t} = 0$, minimalizujeme pouze

$$E(R,t) = \sum_{i=1}^{N_p}\left\|x_i' - Rp_i'\right\|^2. \tag{19}$$

Další úpravou dostaneme

$$E(R,t) = \left(-2\sum_{i=1}^{N_p} x_i'^{\top} Rp_i'\right), \tag{20}$$

což je nutno pro minimalizaci $E(R,t)$ maximalizovat. Dále

$$\sum_{i=1}^{N_p} x_i'^{\top} Rp_i' = tr\left(X'^{\top} RP'\right), \tag{21}$$

kde $tr\left(X'^{\top} RP'\right)$ je stopa matice (trace), což je součet prvků na hlavní diagonále. Ta má vlastnost

$$tr(AB) = tr(BA) \tag{22}$$

pro jakékoliv matice A a B o kompatibilních rozměrech, tedy

$$tr\left(X'^{\top} RP'\right) = tr\left(\left(X'^{\top}\right) RP'\right) = tr\left(RP'X'^{\top}\right). \tag{23}$$

Kovarianční matici o rozměru $d \times d$ označíme $S = PX'^{\top}$ a její jednotkový rozklad(SVD) je

$$S = U\Sigma V^{\top}. \tag{24}$$

Substitucí rozkladu do 23 dostaneme

$$tr\left(RP'X'^{\top}\right) = tr\left(RS\right) = tr\left(RU\Sigma V^{\top}\right) = tr\left(\Sigma V^{\top}RU\right). \tag{25}$$

Jelikož V, R a U jsou ortogonální matice, $M = V^{\top}RU$ je rovněž ortogonální, což znamená, že sloupce M jsou ortonormální vektory a především, že pro každý sloupec Mm_j platí $m_j^{\top}m_j = 1$. Všechy prvky m_{ij} z M tedy mají velikost < 1.

Pro maximalizaci $tr\left(\Sigma M\right)$ využijeme skutečnosti, že Σ je diagonální matice s hodnotami > 0 na diagonále, tedy:

$$tr\left(\Sigma M\right) = \begin{pmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_d \end{pmatrix} \begin{pmatrix} m_{11} & m_{12} & \cdots & m_{1d} \\ m_{21} & m_{22} & \cdots & m_{2d} \\ \vdots & \vdots & \vdots & \vdots \\ m_{d1} & m_{d2} & \cdots & m_{dd} \end{pmatrix} = \sum_{i=1}^{d} \sigma_i m_{ii} < \sum_{i=1}^{d} \sigma_i. \tag{26}$$

Stopa je tedy maximizována, pokud $m_{ii} = 1$. Jelikož M je ortogonální matice, M musí být matice identity a

$$I = M = V^{\top}RU \Rightarrow V = RU \Rightarrow R = VU^{\top}. \tag{27}$$

Po získání rotace lze poté získat translaci minimalizující vzdálenost množin bodů pomocí

$$t = \mu_x + R\mu_p. \tag{28}$$

Určená rotace a translate je dále použita pro získání finální pózy jejím přičtením k původnímu odhadu pózy.

4 Implementace metod

Následující kapitola se zabývá implementací řešení problému, který byl stanoven následně:

- Naimplementovat algoritmus pro detekci a určení pózy předem zvoleného objektu
- Objekt nemá nápadnou texturu
- Detekce probíhá při použití hloubkových a barevných dat - RGB-D
- Detekce musí probíhat v reálném čase, stanovená dolní hranice je 1 FPS
- Výstup musí obsahovat render modelu v detekované póze a pozici
- Demostrovat funkčnost na vhodných datech
- Zhodnotit výsledky zejména s ohledem na robustnost vůči zákrytu

4.1 Použité nástroje

Volba dalších frameworků a knihoven závisí v první řadě na použitém programovacím jazyku. Zde jsem se rozhodl pro C++ ze dvou hlavních důvodů. Prvním z nich je jeho rozšířenost v oblasti počítačového vidění a grafiky, což s sebou přináší mnohem širší možnosti při výběru knihoven pro realizaci kroků algoritmu. První důvod je z velké míry důsledkem toho druhého, což je jeho rychlost, která jej činí vysoce vhodným pro grafiku. Použité IDE bylo Microsoft Visual Studio 2015.

4.1.1 OpenCV

Jedná se o open source knihovnu programovacích funkcí zaměrenou v první řadě na oblast počítačového vidění. Určená je především pro jazyky C a C++, existují však wrappery pro Python, Javu a další. Poskytuje nejrůznější funkcionalitu pro implementaci stanoveného problému, např. načtení obrázku, přečtení hodnoty pixelu a manipulace s ní, konverzi barevného modelu z RGB na HSV, vytvoření 3D point cloudu z hloubkového obrazu, výpočet histogramů nad vstupem a jejich evaluaci na základě různých metrik. Rovněž obsahuje rychlé implementace různých příznaků (SIFT, SURF) a klasifikátorů (SVM, Haar cascade). Nepovinnou součástí knihovny jsou experimentální a příliš neotestované moduly, které je nutno zkompilovat. Jedním z nich je implementace LINEMODu podle [2]. Pomocí OpenCV lze realizovat i rendering výstupního obrazu pomocí modulu Viz3D, který běží na vizualizačním toolkitu VTK.

4.1.2 Point Cloud Library

PCL je další open source knihovnou zaměřenou na počítačovou grafiku, především však na point cloudy, což jsou mračna bodů v 3D prostoru. Obsahuje nejrůznější state-of-the-art algoritmy

na jejich filtrování, výpočet příznaků, rekonstrukci povrchu a segmentaci. Ty lze použít např. pro vizualizaci, filtraci zašumělých dat, spojování několika point cloudů, segmentaci částí scény a extrakci klíčových bodů a výpočet jejich deskriptorů pro rozpoznávání objektů na základě geometrie. Využití má zde díky implementaci algoritmu Iterative Closest Point, který jsem použil pro úpravu finální pózy z počátečního odhadu a vizualiaci této transformace pro vyhodnocení korektnosti.

4.1.3 Blender a Meshlab

Blender je open source řešení pro 3D tvorbu,které v sobě zahrnuje kompletní pipeline - modelování, animace, simulace, rendering, sledování pohybu, editaci videa a dokonce i vlastní herní engine. Dále podporuje skriptování v jazyce Python a možnost instalace různých rozšíření a pluginů, např. podpora importu a exportu dalších formátů objektů, procedurální generování terénu, animovaný text, kalibrace kamery a nespočetně mnoho dalších. Zde je zajímavý pro vytváření LINEMOD templatů z modelu detekovaného objektu. To lze snadno realizovat pomocí animace objektu v jednotlivých pózách a nastavení příslušných parametrů. Příkladem je nastavení zorného úhlu kamery tak, aby odpovídal kameře reálné, zda renderovat barevný objekt nebo jako pixelové hodnoty použít Z-buffer, bitová hloubka a formát výstupu a jiné.

Načítání některých složitých modelů až s miliony vrcholů a povrchů v různých formátech je však obtížné uskutečnit pouze pomocí přidaných pluginů. Tento problém snadno vyřešil poslední použitý program MeshLab, který je rovněž open source a zaměřuje se právě na zpracování a editaci 3D trojúhelníkových sítí (ang. triangular mesh). Na to poskytuje metody pro jejich editaci, zjednodušení, texturování, konverzi mezi formáty a jiné. Oproti Blenderu snadno načetl model libovolného formátu a velikosti, což umožnilo jeho editaci do podoby přijatelnější pro Blender, jelikož LINEMOD díky rozprostření orientací gradientů do okolí nepotřebuje vysoce detailní model pro dosažení korektní detekce.

4.2 Vstupní data

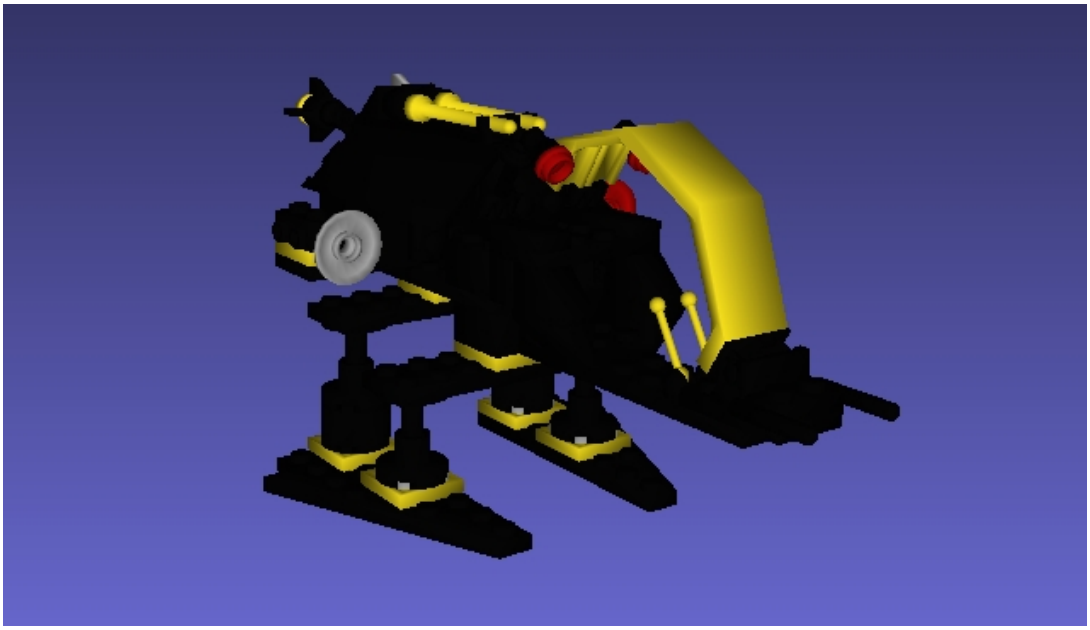
Pro řádné otestování schopností implementovaného přístupu je zapotřebí zvolit nejrůznější typy objektů, které s sebou přinášejí své vlastní výhody, ale i úskalí. Povrch objektů může mít různou strukturu, odlišný počet barev, vysokou podobnost mezi odlišnými úhly pohledu, atd. Rovněž je zapotřebí zvolit vhodný způsob generování templatů tak, aby co nejmenším počtem pokrytých póz bylo stále dosaženo uspokojivých výsledků, jelikož časové nároky LINEMODu stoupají s rostoucím počtem templatů lineárně.

4.2.1 Zvolené objekty

V první řadě jsem zvolil stejný RGB-D dataset použitý v [2] pro srovnání vytvořené pipeline s tou, která byla hlavní inspirací. Dataset se skládá z patnácti převážně jednodušších objektů, které mají zpravidla jedinou barvu na celý objekt a hladký povrch s nepříliš ostrými výškovými

přechody. Neznamená to však, že detekce je podobně jednoduchá, jelikož scény současně obsahují velké množství modelů se složitým pozadím a některé objekty mají mezi sebou díky generalizační schopnosti LINEMODu velkou podobnost. Rovněž obsahuje objekty s nejednoznačnou pózou, kde objekt v určité póze může být identifikován jako mnoho dalších pózách. Dataset rovněž obsahuje ground truth rotace a translace.

Dalším zvoleným objektem je LEGO model Alienator. Ten má složitější povrch, několik barev, ostré hrany a mnoho otvorů. Kvůli lesklosti povrchu však nebylo možné vytvořit dostatečně kvalitní hloubkovou mapu pomocí kamery, testován tedy byl pouze na syntetických sekvencích.



Obrázek 8: Počítačový model Alienator.

Posledním a nejsložitějším objektem je LEGO model bagru, který má ještě větší barevné přechody a jednotlivé součástky plné otvorů. Reálné RGB-D scény jsou dostupné, složitost modelu a nízké rozlišení kamery však vedlo k vysoce nedokonalému zachycení hloubkových dat, kde v porovnání s hloubokovou mapou templatu vygenerovaného z Blenderu je zhruba jen polovina dat.

Obrázek 9: Počítačový model detekovaného bagru.

4.2.2 Generování templatů

Jak bylo výše uvedeno, je potřeba dosáhnout rovnováhy mezi přesnou detekcí a rychlostí algoritmu vygenerováním vhodného množství templatů. Počáteční nastavení byla určena na základě [2]. Pořadí rotace os bylo tedy ZYX (bočení/zatáčení, klopení, klonění, ang. yaw, pitch, roll), pozice kamery v bodě $X = 0$, $Y = 0$, $Z = d$, kde d je počáteční vzdálenost objektu od kamery. Rotace kolem osy a translace proběhly následovně:

- Rotace Z od 180° po 540° (jedna kompletní rotace o 360°), přičemž každý snímek rotoval o 20°
- Rotace Y v rozsahu od 150° po 210° v inkrementech 15°
- Rotace X od 90° po 180° rovněž v 15° inkrementech
- Translace na ose X od $d = 65$ cm do 105 cm, přičemž vzdálenost se navyšuje o 10 cm v každém kroku.

Výsledných templatů je tedy $18 \times 5 \times 7 \times 5 = 3150$. Jeden template se skládá z tří samostatných obrázků, jeden pro barevnou informaci, druhý pro hloubkovou a posledním je maska objektu. Jelikož postprocessing v [2] využívá pouze odstín (hue) a nikolik sytost a jas, stačí pro RGB template zaškrtnout v materiálu příslušného objektu položku shadeless, čímž neproběhne žádné stínování a není zapotřebí zbytečně nastavovat osvětlení. Maska má tam, kde se objekt nachází, bílou barvu s maximální hodnotou, všude jinde maximální černou. Pro tvorbu masky tedy lze

použít stejné nastavení jako pro RGB template, pouze stačí změnit barvu povrchu na bílý. Oba tyto templaty stačí renderovat s 8-bitovou hloubkou, přičemž je nutno nastavit příslušnou barvu, tj. RGB pro barevný template a BW (černobílý obraz, 1 kanál) pro masku. Hloubkový template lze získat pomocí funkce Node Editor a změny Render Layer z Image na Z - výsledkem je hloubková mapa, kde hodnota pixelu značí vzdálenost od kamery v milimetrech. Snadno lze dojít k závěru, že 8-bitová hloubka s 256 hodnotami ani zdaleka nestačí na takový obrázek, zde je tedy použita hloubka 16-bitová s 65536 hodnotami a jedním barevným kanálem. Maximální hodnotu Blender dává rovněž tam, kde žádný objekt není (bod je nekonečně daleko), na první pohled tedy hloubkový obraz vypadá jako invertovaná maska.

Obrázek 10: Trojice renderovaných obrázků tvořící template pro objekt duck, zleva RGB, hloubková mapa, maska. Hloubková mapa byla pro lepší viditelnost renderována jako 8-bitový obraz.

4.3 Detekční pipeline

V následující podkapitole budou více přiblíženy použité metody z programového hlediska a samotný chod detekční pipeline.

4.3.1 LINEMOD v OpenCV

Jak již jsem se zmínil, knihovna OpenCV implementaci LINEMODu obsahuje v modulu rgbd, který je v nepovinné součásti opencv_contrib, kterou je nutno zkompilovat. Ta v sobě obsahuje veškerou funkcionalitu od zpracování templatů včetně preprocessingu až po získání potenciálních oblastí v obraze, kde by se objekt mohl identifikovat. Prvním krokem je přidání templatů do objektu `Detector`, což obstarává metoda

```
int addTemplate ( const std::vector<Mat> & sources, const String & class_id,
const Mat & object_mask, Rect *bounding_box=NULL )
```

kde sources je pole templatů, každý pro jednu modalitu (tedy jeden pro RGB a jeden pro D) a class_id určuje, ke které třídě objektů daný template patří, což je v našem případě detekce jediného objektu 1. Dále object_mask je maska objektu pro určení oblasti, kterou má hledat a bounding_box je nepovinný parametr, který umožňuje zmenšit template na danou podoblast.

Ten je nastaven na tu stejnou, která byla určena při sestavení konfiguračního souboru pro daný objekt.

Další nepostradatelnou funkcionalitou je zde možnost uložení spočítaných příznaků do souboru formátu .yml a jejich načtení při opakovaném chodu, což značně urychlí práci, jelikož zpracování templatů do příznaků trvá při 3150 vzorech několik minut. Oproti tomu načtení ze souboru je provedeno za několik sekund. To zprostředkovávají funkce

`void cv::linemod::Detector::read ( const FileNode & fn )` a

`void cv::linemod::Detector::write ( FileStorage & fs ) const`

pro načtení celého souboru a `readClass` a `writeClass` pro načtení pouze určité třídy. Samotnou detekci pak spouští metoda

```
void cv::linemod::Detector::match ( const std::vector<Mat> & sources,
float threshold, std::vector<Match> & matches, const std::vector<String>
& class_ids = std::vector<String>(), OutputArrayOfArrays quantized_images =
noArray(), const std::vector<Mat> & masks = std::vector<Mat>() ) const
```

kde `sources` je pole obsahující modality scény, ve kterých je objekt detekován, `threshold` je minimální skóre detekcí, které detektor vrátí, `matches` je samotný výsledek detekce, `class_ids` určuje čísla detekovaných tříd, `quantized_images` vrací kvantizované obrazy vstupu, a `masks` určuje podoblast, ve které má detektor vyhledávat detekce. V našem případě detekce v celém obrazu masku nepoužíváme. Nejpodstatnější parametr je zde `matches`, což je pole všech nalezených detekcí (tedy těch, které byly rovno nebo větší `threshold`, seřazených podle hodnoty podobnosti. Samotný objekt `Match` má pět parametrů: `String class_id` udává třídu nalezeného objektu, `float` similarity je hodnota podobnosti od 1 (100%) po 0, `int template_id` značí, který template tuto detekci nalezl, a `int x` a `int y` udávají levý horní roh bounding boxu této nalezené oblasti.

Po vyzkoušení je však patrné, že sám o sobě tento detektor nelze pro detekci použít, jelikož z barevných informací používá pouze gradienty a u hloubkové mapy pouze orientace, nikoliv samotnou velikost hodnoty, navíc obě modality diskretizuje do několika hodnot. Výsledkem hledání jediného objektu i v relativně jednoduché scéně má za výsledek desítky až stovky potenciálních detekcí, z nichž je valná většina falešná. Nejjednodušší možností je vzít tu s největší podobností, často se však jedná o zcela jiný prvek než ten, který byl hledaný. Je teda nutno pro určení finálního templatu a jeho pozice v obraze použít několika metod postprocessingu. Jedná se však o velice rychlý inicializátor detekce, který dokáže při zhruba 3000 vzorech dosáhnout 10 FPS na Intel Core i5-2320 CPU 3,00 GHz se čtyřmi jádry.

4.3.2 Postprocessing

Po získání všech potenciálních korektních detekcí je potřeba vybrat jedinou správnou. To lze provést další analýzou společných vlastností detekovaného místa a templatu, který tam byl detekován. Lze pro to zde využít:

- Výstupní hodnotu LINEMODu – míra podobnosti, automaticky seřazená od největší
- RGB, popř. HSV informace – porovnávání na základě barvy
- Hloubkové mapy
- Vzdálenost mezi point cloudy detekovaného objektu a místa, kde byla detekce nalezena

Zde je výhodné použití kaskádového přístupu, kde se nejprve přistupuje k výpočetně nenáročným metodám pro prvotní eliminaci nekorektních detekcí, což snižuje čas chodu. Nejjednodušší z nich je triviální accept/reject na základě hloubky středového pixelu, kde projde testem pouze template, kde je rozdíl hloubek ve středu detekované oblasti a templatu menší než daná prahová hodnota. Nelze však tento přístup použít pro objekty s náhlými hloubkovými změnami nebo otvory. Dále jsem využil dvou přístupů navrhnutých v [2], z nichž první je porovnávání na základě barevného odstínu (hue) detekované oblasti. To činí porovnání robustní vůči změnám osvětlení a jiným vlivům. Konverzi mezi barevnými modely lze v OpenCV lze snadno provést, hodnota odstínu H se poté namapuje v rozmezí 0 až 180. Obrazy se poté porovnávají pixel po pixelu tam, kde maska templatu značí existenci objektu a výstupní hodnota je procento párů pixelů, které měly na dané pozici rozdíl odstínů menší než zadanou prahovou hodnotu. Stejným jednoduchým způsobem lze porovnat hloubku jako procento hloubek pixelů s akceptovatelným rozdílem hloubkových hodnot nebo průměrný hloubkový rozdíl párů bodů, který nesmí přesáhnout stanovený práh. Obě tyto metody jsou triviálně implementovatelné díky generování syntetických templatů z počítačového modelu, což umožňuje snadné získání masky a separace objektu od pozadí. Nevýhoda těchto point-by-point metod je však patrná u složitějších objektů nebo objektů s mnoha barvami, kde i drobný posun nebo nepřesnost ve snímaných datech má za důsledek vysoký pokles podobnosti. Zde je poté nutno přistoupit k metodám, které nejsou tolik citlivé na drobný posun nebo u takových objektů nastavit prahové hodnoty níže. Další možností, která byla použita v [2] je provedení algoritmu ICP s nízkým počtem iterací a rozhodnutí o korektnosti detekce na základě vzdálenosti korespondujících bodů, při použití současné implementace tohoto algoritmu však často docházelo k nízkým vzdálenostem i v případě zcela nekorektních detekcí, jako finální detekci jsem tedy považoval template s nejvyšší podobností LINEMODu který současně prošel všemi testy postprocessingu.

4.3.3 Úprava pózy algoritmem ICP

Výstupem z LINEMODu je jediný template objektu, který prošel detekcí a všemi kroky postprocessingu. Ten však udává pouze hrubý odhad pózy objektu, který je nutno dále upravit tak,

aby se co nejvíce podobal reálnému obrazu. Pro provedení tohoto kroku je nejprve nutno získat vhodnou 3D reprezentaci objektu pro template i reálný obraz, což je point cloud – mrak bodů v prostoru, které reprezentují místa na objektu. Pro template se zde opět prokazuje výhoda generování syntetických templatů, jelikož stejný model ze využít k vytvoření point cloudu, který je potřeba pouze rotovat do stejné pózy jako detekovaný template. V případě vstupního obrazu je situace složitější, zde je nutno vytvořit point cloud z hloubkové mapy scény. Pro to lze použít metodu

```
void cv::rgbd::depthTo3d (InputArray depth, InputArray K,
OutputArray points3d, InputArray mask=noArray())
```

z OpenCV, kde `depth` je vstupní hloubková mapa, `K` je kalibrační matice použité kamery, `points3d` je výstupní pole 3D bodů a `mask` je maska určující body, které se mají v point cloudu vyskytovat. Zde lze opět použít masku templatu, jelikož ve správné detekci by objekty měly být vysoce podobné a případě vybrání zcela nekorektního templatu uvízne ICP v lokálním minimu nezávisle na použití špatné masky. V případě převodu detekované oblasti bez masky výsledný point cloud obsahuje veškeré pozadí, což komplikuje nalezení korespondujících bodů. Pro práci s 3D reprezentací objektu jsem použil knihovnu Point Cloud Library (PCL), která v sobě implementaci Iterative Closest Point obsahuje. Prvním krokem po vytvoření detektoru je nastavení vstupního a cílového cloudu pomocí

`setInputCloud (const PointCloudInConstPtr &cloud)` a
`setTargetCloud (const PointCloudInConstPtr &cloud)`,

kde rotace probíhá na vstupním cloudu tak, aby se minimalizovala vzdálenost korespondujících bodů mezi cloudy. Dále lze nastavit nejrůznější parametry, např. maximální vzdálenost mezi korespondujícími body, maximální počet iterací a minimální hodnotu epsilon (velikost změny mezi iteracemi) při které pokračovat v chodu a další. Transformace se poté spouští metodou `align (PointCloudSource &output)`, kde `output` je výsledný point cloud a samotnou transformační matici lze získat pomocí `getFinalTransformation()`. Finální rotace je pak získána odečtením rotace z transformační matice od původního odhadu rotace.

4.3.4 Popis pipeline

K chodu je zapotřebí nastavení vstupních dat, která se nacházejí ve složce *data* a v ní ve složce s názvem detekovaného objektu. Zde se v první řadě nachází složky *render_rgb*, *render_depth* a *render_mask*, které obsahují veškeré templaty. Ty jsou pojmenovány číselně včetně nul s příponou .png, např. u 3150 templatů je číslování od *0000.png* po *3150.png*. Dále jsou zde složky *detect_rgb* a *detect_depth*, které obsahují sekvence barevných a hloubkových snímků, ve kterých se objekt detekuje. Ty mají stejný formát jako dataset [2], kde barevné snímky jsou ve formátu

JPG s jmény od *color0* po *n-1* pro n snímků a hloubkové formátu PNG a pojmenovány od *depth0*. Dále je zde zapotřebí point cloud, který je ve formátu .txt se jménem *pointcloud*, první řádek značí počet bodů a každý další řádek je trojice souřadnic bodu. Model pro vizualizaci výsledku je ve formátu .ply a pojmenován *plymodel*. Konfigurace templatů je v souboru *templates.txt* a vypočítané příznaky pro znovupoužití se ukládají do souboru *templates.yml*. Výsledné vizualizace se nakonec ukládají do složky *output*. V případě použití ground truth póz se nacházejí ve složce *positions*.

Prvním krokem je určení detekovaného objektu (tedy pracovní složky), zda generovat konfiguraci templatů a zda templaty jsou z .yml souboru nebo jestli se mají generovat. První řádek konfigurace templatů je číslo posledního templatu bez přípony, každý řádek je poté pro jeden template v následujícím pořadí: název souboru, souřadnice x bounding boxu, souřadnice y bounding boxu, šířka bounding boxu, výška bounding boxu, rotace kolem osy Z, rotace kolem osy X, rotace kolem osy Y, vzdálenost od kamery v centimetrech. Po načtení konfigurací a templatů, ať už ze souboru nebo z obrázků, zpracuje program každý vstupní snímek. Nejprve jej načte do detektoru LINEMOD, provede nad ním detekci, vrátí seznam potenciálních detekcí, nad těmi se poté provede postprocessing, přičemž ten první, který projde postprocessingem (tedy ten s největším skóre z LINEMODu, jelikož výsledky jsou seřazené) má dále upravenou pózu pomocí modelu objektu a point cloudu vypočítaného ze vstupního snímku, čímž se určí finální rotace. Model objektu je renderován ve vstupním obraze na detekovaném místě ve finální póze a obrázek se nakonec uloží do výstupní složky.

5 Evaluace výsledků

Tato kapitola se věnuje dosaženým výsledkům z hlediska rychlosti detekce, přesnosti a odolnosti vůči změnám vlastností detekované scény a objektu.

5.1 Rychlost detekce

Jako cílová hranice byla stanovena rychlost 1 FPS, přičemž rychlost detekce ovlivňuje řada faktorů. Nejdůležitějším z nich je nastavení prahové hodnoty pro detekce LINEMODu, nad kterou se detekce považuje za potenciálně korektní. Tu je pro rychlou a zároveň korektní detekci zapotřebí nastavit na vhodnou hodnotu, která je určená povahou scény a zachytávaných dat. Prvním příkladem je dataset [2], kde má scéna velké množství dat v pozadí, objekt je malých rozměrů a dobře viditelný. Korektní template v tomto případě zpravidla dosahuje podobností nad 0,9. Při nastavení hranice na 0,85 LINEMOD potřebuje přibližně 0,2 vteřiny pro jednu detekci, přičemž potenciálních detekcí je několik tisíc. Snížení hranice na 0,5 spolu s jednoduchým objektem má za výsledek počet potenciálních detekcí v řádu statisíců a rychlost již poklesne na 10 vteřin na jednu detekci, což je způsobeno jednoduchou strukturou objektu, kterou při tak nízké prahové hodnotě lze snadno nalézt na pozadí s velkým množstvím objektů. Opačným případem je detekce podstatně složitějšího a většího modelu alienator v zhruba 20% zákrytu na prázdném pozadí, který má podobnost přibližně 0,7. V případě použití prahové hodnoty 0,5 na tomto místě dojde k detekci 1500 až 2000 instancí, přičemž jedna detekce trvá zhruba 0,35 vteřin.

Obrázek 11: Příklad výsledku korektní detekce modelu Ape z datasetu [2].

Druhým nastavitelným faktorem, který potenciálně ovlivňuje rychlost detekce je počet iterací algoritmu Iterative Closest Point. Rotace vzoru je však v případě správné detekce velmi blízko póze detekovaného objektu a algoritmus se ukončí již po několika iteracích. Samotný proces ICP trvá 0,1 až 0,2 vteřin. Čas postprocessingu je zanedbatelný, jelikož se zde jedná pouze o porovnávání pixelů na stejných pozicích.

Za použití procesoru Intel Core i5-2320 CPU 3,00 GHz se čtyřmi jádry trval celý proces od načtení vstupu po vykreslení modelu v dané póze v průměru 0,75 vteřiny pro jeden snímek na datasetu [2], na ostatních dvou 1 FPS, stanovená hranice tedy byla dosažena.

5.2 Srovnání s algoritmem LINE-2D

LINE-2D je obdoba algoritmu LINEMOD, která využívá pouze gradienty barev. Následující test proběhl na čtyřech objektech datasetu [2], přičemž měřeny byly odchylky rotace a translace objektu od ground truth hodnot a celkový čas chodu detekce pro sekvenci 1235 snímků ve vteřinách T. Odchylka translace D_t je průměr odchylek ze všech snímků, která je pro jeden snímek určena jako součet vzdáleností od ground truth na ose X a Y v pixelech. Chyba rotace D_r je rovněž průměr odchylek rotace ze všech snímků, přičemž odchylka rotace od grouth truth je určena jako průměr odchylek od ground truth na osách X, Y a Z v radiánech. V obou případech byly použity stejné parametry a metody postprocessingu. Prahová hodnota LINE algoritmů byla nastavena na 0,85, rozdíl odstínů pixelů na 15 a počet platných pixelů pro přijmutí detekce na základě barvy na 75%.

Tabulka 1: Porovnání algoritmů LINEMOD a LINE-2D z hlediska rychlosti a chyby translace a rotace.

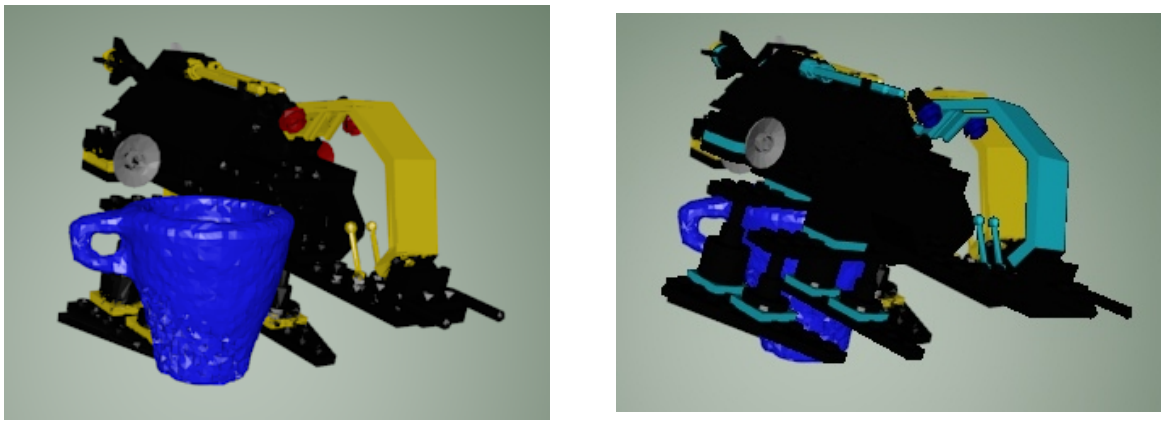
	LINE-2D			LINEMOD		
Objekt	T	D_t	D_r	T	D_t	D_r
Ape	$970,9$	$64,6$	$0,34$	929,4	$36,7$	$0,14$
Duck	$830,3$	$80,2$	$0,41$	866,5	$47,5$	$0,16$
Cup	$741,6$	$59,6$	$0,56$	947,8	$40,2$	$0,35$
Holepuncher	$1215,2$	$141,7$	$0,58$	857,7	$73,5$	$0,19$

Hodnoty D_t jsou čistě orientační, jelikož poskytnuté pozice objektů [2] byly určeny jinou metodou než v případě zde implementované pipeline, kde za pozici objektu v obraze považujeme střed výsledného bounding boxu. Z výsledků je však patrný velký nárůst přesnosti odhadu translace i rotace v případě použití hloubkových dat společně s barevnou informací. Nelze však jednoznačně určit, který přístup je rychlejší, jelikož záleží na počtu a správnosti detekcí, které jsou výstupem příslušného algoritmu LINE. Zpomalení nastává především v případech, kde je nutno zkontrolovat mnoho detekcí před nalezením té, která projde všemi kontrolami postprocessingu.

5.3 Odolnost vůči zákrytu

Další důležitou vlastností je chování algoritmu v případě zakrytí částí objektu jiným objektem. Pro tento účel byl zde použit model alienator spolu s počítačově generovanými scénami, jelikož tak lze snadno vytvořit situaci v zákrytu a mimo zákryt pro přesné porovnání. Překrytí ovlivňuje

všechny části pipeline, lze tedy očekávat, že v případě nalezení správného vzoru bude podobnost znatelně nižší, rovněž tak u postprocessingu bude mít menší část plochy očekávanou barvu nebo hloubku. V případě algoritmu Iterative Closest Point rovněž buď dojde k extrakci překrývajícího objektu nebo jeho vyfiltrování na základě podobnosti s vzorovým obrazem. Zde však nelze zaručit, že překrývající objekt je těsně před detekovaným objektem, což bez problémů projde hloubkovou kontrolou, nebo jestli nemá stejnou barvu. K takovému případu může snadno dojít u jednobarevných objektů vzhledem k tomu, že pro porovnání je použitý pouze odstín.



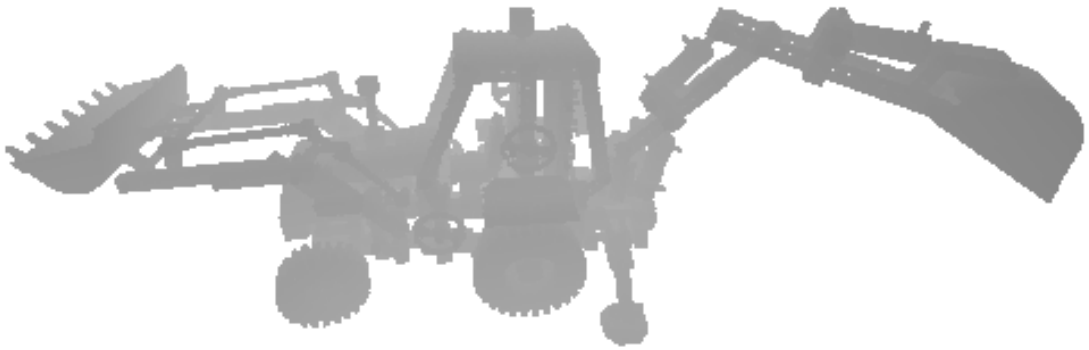
Obrázek 12: Detekce modelu Alienator při částečném zakrytí jiným objektem, zobrazený model má odlišnou barvu pro lepší viditelnost výsledku.

Scény byly renderovány oproti vzorům za použití světla a stínování, přičemž u scén bez zákrytu se podobnost LINEMODu pohybovala kolem 0,85, zatímco počet pixelů se správnou barvou byl přibližně 78%. U stejné scény se zákrytem došlo k snížení barevné podobnosti např. na pouhých 61%, u LINEMODu se podobnost však v některých případech vůbec nezměnila. K tomu došlo především v situacích, kde byl model zakrytý v jeho těsné blízkosti, zatímco v jiných případech, kde k zakrytí došlo 20 cm před objektem, byla podobnost o něco nižší navzdory zakrytí menší části detekovaného objektu. Největší hrozbou je zde nutnost snížení prahových hodnot, což vzhledem k množství potenciálních detekcí snadno vede k nekorektním výsledkům, kde objekt v pozadí dosáhne jedné z nejvyšších hodnot podobnosti LINEMODu současně s validní hloubkou a barvou. Zakrytý objekt zde mezitím o několik procent přijde, což vysoce ztěžuje jeho korektní detekci.

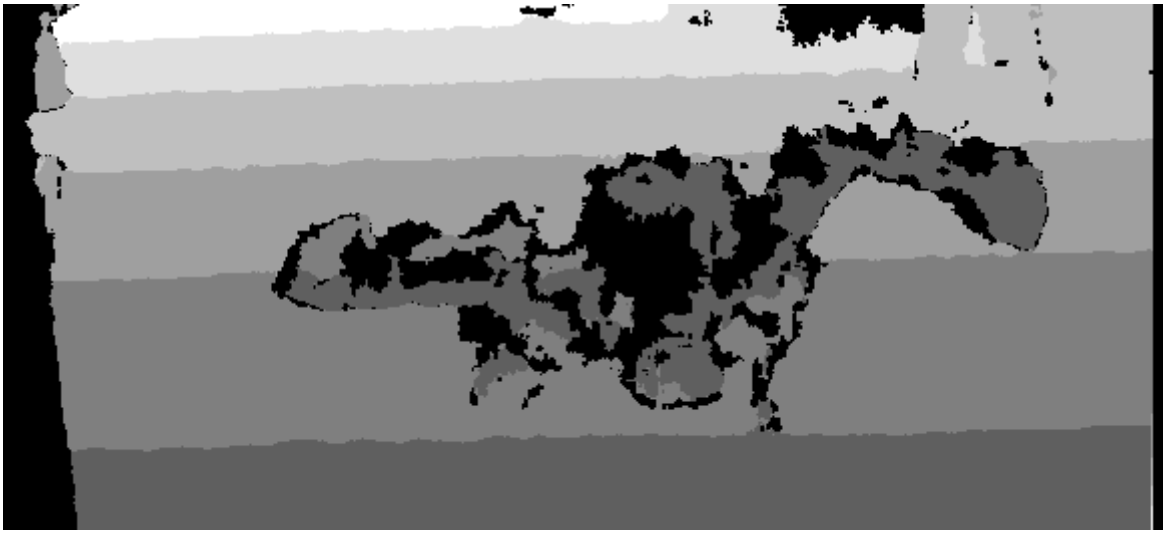
5.4 Detekce složitého objektu se špatnou kvalitou zdrojových dat

V následujícím testu bylo cílem vyzkoušet, jak pipeline funguje v případě nepřítomnosti většiny přívětivých vlastností objektů v datasetu [2]. Výše zmíněný model bagru je několikanásobně větší a má více barev, což podstatně komplikuje porovnávání odstínu podle jednotlivých pixelů. Dále má mnoho otvorů a podlouhlých částí, což činí složitějším porovnávání i na základě hloubky. Největším problémem je zde však špatná kvalita vstupního hloubkového obrazu, který se syn-

teticky generovanému modelu pouze zhruba podobá tvarem. Korektní výstupy LINEMODu pro tento objekt měly podobnost přibližně 0,72 a při zvolené prahové hodnotě 0,65 trvalo zpracování jednoho snímku v průměru 0,37 vteřiny. Postprocessing na základě odstínu je i v tomto případě stále použitelný, je však nutno změnit parametry. Přijatelný rozdíl odstínu pixelů byl zde nastaven na 0,3 a prahová hodnota procentuální shody pixelů na 0,65. Je však nutno podotknout, že objekt byl umístěn v místnosti s poměrně konstantním pozadím o jediné barvě, což tento krok pravděpodobně značně ulehčilo. Oproti tomu je hloubková kontrola na základě porovnání pixelů nepoužitelná, jelikož zhruba polovina pixelů nacházející se na masce templatu má nekorektní hodnotu blízkou nule. To rovněž komplikuje odhad vzdálenosti reálného objektu, protože je nutno rozlišit místa, kde je dálková hodnota korektní a kde ne. Stejně tak ovlivňuje kvalita hloubkových dat i algoritmus Iterative Closest Point, kde nekorektní a chybějící hloubkové údaje mohou vést k zhoršení finálního odhadu pózy místo jeho zlepšení.



Obrázek 13: Počítačem generovaná hloubková mapa modelu bagru. Pro lepší viditelnost dat je zde uložena do 8-bitového obrazu.



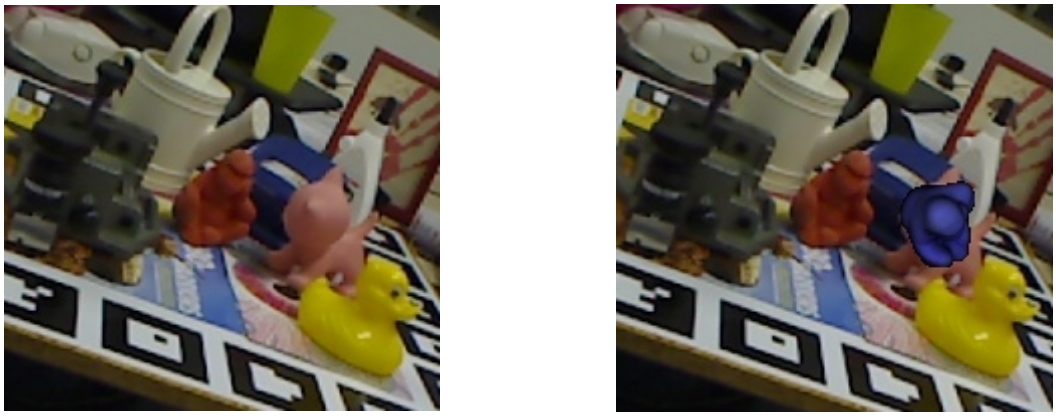
Obrázek 14: Hloubková mapa objektu bagru, oproti syntetické je zde podstatně nižší kvalita a počet dat.

Obrázek 15: Výsledek detekce modelu bagru, zobrazený model má odlišnou barvu pro lepší viditelnost výsledku. Na výsledku je patrný nepřesný odhad pózy a vzdálenosti objektu kvůli složitosti objektu a neúplným hloubkovým datům.

5.5 Problémové situace

Jak již bylo výše uvedeno, valná většina chyb detekce je případ false positive, tedy detekce byla nalezena, ale na špatném místě nebo ve špatné póze a nastanou, když nekorektní vzorový obraz má silnou podobnost s některým prvkem pozadí včetně hloubky a barevného odstínu. Dobrým

příkladem jsou objekty ape a cat v datasetu [2], kde v určitých pózách dochází k záměně objektů, navíc mají téměr identický odstín barvy a stejnou velikost. Pokus o rozšíření barevné kontroly na více složek (odstín a sytost) měl za výsledek značný pokles celkové přesnosti detekce. Další problémovou situací je případ, kde template zachycuje pouze část objektu, příkladem je např. model hrnku s uchem v datasetu [2]. V případě, že template má ucho za objektem tedy snadno dojde k detekci v póze, kde ucho vyčnívá zcela nalevo nebo napravo. Extrémním případem jsou dále objekty s pravidelnou strukturou, kde lze jednu pózu korektně interpretovat jako mnoho dalších, např. u koulí nebo nádob bez ucha a podobných rozlišujících prvků. V [2] byla pro tyto typy objektů použita jiná metrika vyhodnocení, která je méně citlivá na velké rozdíly mezi detekovanou a korektní rotací na jedné ose.



Obrázek 16: Případ nekorektní detekce, kde objekt Cat je detekován jako objekt Ape.

Lze rovněž snadno na základě předchozích poznatků vyvodit, že při kombinaci složitého pozadí s jednoduchým detekovaným objektem a přítomností značného zákrytu se může LINEMOD stát téměř nepoužitelným, jelikož je zapotřebí snížení prahové hodnoty pro LINEMOD. To má za důsledek velký pokles rychlosti a velký nárůst potenciálních detekcí, přičemž ta správná se může nacházet za několika tisíci nekorektních. Současně je nutno učinit méně přísným i postprocessing, což rovněž zvyšuje pravděpodobnost přijmutí nesprávného vzoru.

6 Závěr

Cílem mé diplomové práce bylo zvolení a implementace algoritmu detekce jednoho konkrétního objektu v obraze, přičemž je nutno rozpoznat nejen umístění, ale i pózu objektu. Detekovaný objekt je netexturovaný a detekce probíhá na základě barvy a hloubkové mapy. Dalším požadavkem detekce byl chod v reálném čase, přičemž za dolní hranici požadujeme 1 FPS. Výstup současně musí mít grafické zobrazení, kde je detekovaný objekt vykreslen na detekovaném místě v příslušné póze. Výběr proběhl na základě analýzy současného stavu, kde jsem zmínil aktuální metody pro řešení problému detekce objektů. Ne všechny však splňují stanovené požadavky, hlavním problémem zde byla rychlost algoritmů, která často byla pod stanovenou hranicí. Druhým problémem byly hardwarové požadavky, jelikož mnoho přístupů dosahuje žádané rychlosti díky výpočtům na vysoce výkonných a drahých počítačových sestavách, které jsem zde neměl k dispozici. Nakonec jsem se rozhodl pro řešení na základě algoritmu LINEMOD [1, 2], jelikož je osvědčený, dobře zdokumentovaný a dostatečně rychlý i na méně výkonných počítačích. Implementaci jsem provedl v programovacím jazyce C++ za použití knihoven Point Cloud Library pro práci s 3D daty a OpenCV, která obsahuje mnoho možností pro zpracování obrazu a manipulaci s jeho daty včetně implementace algoritmu LINEMOD. Pro práci s modely detekovaných objektů a generování vzorových obrazů jsem použil programy Blender a Meshlab.

Celkový chod detekční pipeline byl inspirován především postupem v práci [2], kde LINEMOD slouží jako počáteční hrubý odhad pózy objektu, který je dále upraven algoritmem Iterative Closest Point a platnost detekce kontrolována na základě porovnání hloubkových a barevných dat detekovaného a zdrojového obrazu. Při testu vytvořené pipeline jsem se soustředil zejména na dva objekty, které v porovnání s objekty datasetu použitém v [2] mají větší rozměry, složitější povrch a více barev. První z nich, alienator, byl testován na počítačem vytvořených scénách za přítomnosti částečného zákrytu, druhým modelem byl model LEGO bagru v reálné scéně, kde došlo k neúplnému zachycení hloubkových dat. Na základě testů jsem zjistil, že odolnost testovaného přístupu vůči zákrytu závisí kromě velikosti zakryté části objektu především na složitosti pozadí scény a složitosti detekovaného objektu, přičemž jednodušší objekty jsou více náchylné na nekorektní detekce. U pozadí naopak usnadňuje detekci malé množství rušivých prvků a odlišných barev. V případě neúplných hloubkových dat dosáhl LINEMOD překvapivě dobrých výsledků, použité metody postprocessingu na základě hloubky však u takových vstupních dat ztratily svou účinnost. Rychlost detekce jednoho snímku se pohybovala v rozmezí od 0,6 po 1 vteřinu v závislosti na detekované scéně, složitosti objektu a stanovených prahových hodnotách, požadavek na rychlost byl tedy splněn. Provedeno bylo rovněž porovnání metody LINEMOD, která využívá hloubku i barvu s metodou LINE-2D, která využívá pouze gradienty barev. Zde bylo prokázáno znatelné zlepšení v případě použití obou modalit.

K obtížným částem práce patřila především volba vhodné metody kvůli zadaným požadavkům, které značně zúžily výběr použitelných metod. Další náročnou částí byla práce s experimentálními knihovnami v OpenCV, zejména v kombinaci s jinými open source prostředky jako

Point Cloud Library, kde při současném použití obou knihoven často docházelo k pádu aplikace. Rovněž docházelo v případě PCL k nefunkčnosti některých metod, například při konverzi bodů v prostoru z objektu OpenCV do PCL, což bylo nutno vyřešit vlastní konverzí.

Největší možnost pro další zlepšení pipeline pravděpodobně spočívá v použití jiného přístupu při algoritmu Iterative Closest Point a v případě složitých objektů jiných metod postprocessingu, které nefungují na principu porovnávání jednotlivých bodů. Současně je však potřeba se stále držet stanovených časových požadavků, což by mohlo být téma další práce.

Literatura

[1] Hinterstoisser, S., et al. Gradient Response Maps for Real-Time Detection of Texture-Less Objects, TPAMI 2012.

[2] Hinterstoisser, S., et al. Model Based Training, Detection and Pose Estimation of Texture-Less 3D Objects in Heavily Cluttered Scenes, ACCV 2012.

[3] Hsiao, E., Hebert, M. Gradient Networks: Explicit Shape Matching Without Extracting Edges. AAAI 2013.

[4] Hsiao, E., Hebert, M. Occlusion Reasoning for Object Detection under Arbitrary Viewpoint. CVPR 2012.

[5] Zhe Cao, Yaser Sheikh, Natasha K. Banerjee. Real-time scalable 6DOF pose estimation for textureless objects. ICRA 2016.

[6] Tejani, A., et al. Latent-Class Hough Forests for 6 DoF Object Pose Estimation. PAMI 2016.

[7] Fernández-Villaverde, J. "Kalman and particle filtering."The New Palgrave Dictionary of Economics. Second Edition. Eds. Steven N. Durlauf and Lawrence E. Blume. Palgrave Macmillan, 2008. The New Palgrave Dictionary of Economics Online. Palgrave Macmillan.

[8] Changhyun Choi, Henrik I. Christensen. RGB-D Object Tracking: A Particle Filter Approach on GPU. Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Tokyo Big Sight, Japan, 2013.

[9] Julier, SJ, Uhlmann, JK. A New Extension of the Kalman Filter to Nonlinear Systems. SPIE 1997.

[10] Scaramuzza Davide. Tutorial on Event-based Vision for High-Speed Robotics. IROS 2015.

[11] Gallego G., E.A. Lund, J., Mueggler, E. et al. Event-based, 6-DoF Camera Tracking from Photometric Depth Maps. PAMI 2017.

[12] Brachmann Eric, Krull Alexander, et al. Learning 6D Object Pose Estimation using 3D Object Coordinates. ECCV 2014.

[13] Drost, B., et al. Model Globally, Match Locally: Efficient and Robust 3D Object Recognition. IEEE 2010.

[14] Shotton, J., Glocker, B., Zach, C., Izadi, S., Criminisi, A., Fitzgibbon, A. Scene coordinate regression forests for camera relocalization in rgb-d images. CVPR 2013.

[15] Bonde, U., Badrinarayanan, V., Cipolla, R. Robust Instance Recognition in Presence of Occlusion and Clutter. ECCV 2014.

[16] L. Bo, X. Ren, D. Fox. Unsupervised Feature Learning for RGB-D Based Object Recognition. ISER 2012.

[17] L. Bo, X. Ren, D. Fox. Hierarchical Matching Pursuit for Image Classification: Architecture and Fast Algorithms. NIPS 2011.

[18] Holzer, S., Shotton, J., Kohli, P. Learning to efficiently detect repeatable interest points in depth data. ECCV 2012.

[19] Steger, C. Occlusion Clutter, and Illumination Invariant Object Recognition. International Archives of Photogrammetry and Remote Sensing 2002.

[20] Payet, N., Todorovic, S. From Contours to 3D Object Detection and Pose Estimation. ICCV 2011.

[21] Liebelt, J., Schmid, C. Multi-View Object Class Detection with a 3D Geometric Model. CVPR 2010.

[22] Tola, E., Lepetit, V., Fua, P. DAISY: An Efficient Dense Descriptor Applied to Wide Baseline Stereo. IEEE 2009.

[23] Rios-Cabrera, R., Tuytelaars, T. Discriminatively Trained Templates for 3D Object Detection: A Real Time Scalable Approach. ICCV 2013.

[24] Kehl, Wadim, et al. Hashmod: A Hashing Method for Scalable 3D Object Detection. BMVC 2015.

[25] Hajiaghayi, M.T. Iterative Closest (Corresponding) Point (ICP) Algorithms. Massachusetts Institute of Technology, 2005. Dostupné z: http://groups.csail.mit.edu/graphics/classes/6.838/F01/lectures/IterativeAlgs /ICP/present.html

[26] Rusinkiewicz, S., Levoy, M. Efficient Variants of the ICP Algorithm. 3DIM 2001.

[27] Pavlakos, G., Zhou, X., Chan, A., Konstantinos, D. a Kostas, D. 6-DoF Object Pose from Semantic Keypoints. ICRA 2017 [online]. Dostupné z: https://arxiv.org/pdf/1703.04670.pdf

[28] Tombari, F., Salti, S., Di Stefano, L. Unique Signatures of Histograms for Local Surface Description. ECCV 2010 [online]. Dostupné z: https://vision.deis.unibo.it/fede/papers/eccv10.pdf

[29] Tombari, F., Salti, S., Di Stefano, L. A combined texture-shape descriptor for enhanced 3D feature matching. ICIP 2011 [online]. Dostupné z: https://www.vision.deis.unibo.it/fede/papers/icip11.pdf

[30] G. Lowe, D. Distinctive Image Features from Scale-Invariant Keypoints. IJCV 2004 [online]. Dostupné z: https://people.eecs.berkeley.edu/ malik/cs294/lowe-ijcv04.pdf

[31] Li, C., Bohren, J., Carlson, E., Hager, G. Hierarchical Semantic Parsing for Object Pose Estimation in Densely Cluttered Scenes. ICRA 2016 [online]. Dostupné z: https://www.cs.jhu.edu/ cli53/papers/chi_icra16.pdf

[32] Schwarz, M., Schulz, H., Behnke, S. RGB-D Object Recognition and Pose Estimation based on Pre-trained Convolutional Neural Network Features. ICRA 2015.

[33] Rusu, R. B. Semantic 3d object maps for everyday manipulation in human living environments. KI-Knstliche Intelligenz, 2010.

[34] Bay, H., Tuytelaars, T., Van Gool, L. Surf: Speeded Up Robust Features. ECCV 2006 [online]. Dostupné z: http://www.vision.ee.ethz.ch/ surf/eccv06.pdf

A Příloha CD/DVD

DVD je zaheslováno, přístupové heslo je A1234567.

- linemod_detection.cpp - zdrojový kód pipeline
- data.rar - veškerá data potřebná pro běh pipeline, vzorové obrazy jsou již vyrenderovány, obsažené objekty jsou Ape, Duck, Holepuncher, Cup, Backhoe a Alienator